


Centre Microélectronique de Provence

Cycle Ingénieur de l'Ecole Nationale Supérieure des Mines de Saint-Etienne Spécialité Electronique et Informatique Industrielle




# Projet de fin d'études



## CONCEPTION ET REALISATION D'UN DATALOGGER DE TEMPERATURE RFID.

Présenté par
**Rudy HOUQUE**

**Le 16 juin 2010**

Société
**STMicroelectronics**

**PRESIDENT DU JURY :** **Christophe HAMMAN** (SOLLAC)
**MEMBRES DU JURY :** **François JAUJARD** (EMSE-SE)
**Laurent FREUND** (EMSE-SE)
**Jean-Marie GAULTIER** (STMicroelectronics)









Je souhaite remercier l'ensemble des acteurs qui m'ont permis de suivre cette formation et de la mener à bien et en particulier :

- La société STMicroelectronics qui m'a offert l'opportunité d'effectuer mon apprentissage dans des conditions idéales.
- Monsieur Jean-Marie Gaultier, mon tuteur industriel et supérieur hiérarchique, pour son encadrement et sa pédagogie.
- Monsieur Emmanuel Boulet, technicien de l'équipe application, pour son expérience et son savoir qu'il a su partager.
- Monsieur Hugues Creusy ingénieur de l'équipe d'application, pour ses conseils et le partage de ses connaissances.
- Monsieur Pascal Castanet ingénieur d'application, pour son aide et sa disponibilité.
- Monsieur Sylvain Fidelis directeur marketing, pour sa motivation et son sens de l'efficacité.
- Monsieur Christophe Mani mon ancien tuteur industriel pour l'attention qu'il a porté à son rôle de tuteur et la rapidité avec laquelle il m'a aidé à m'intégrer dans l'entreprise.
- Monsieur Jean-Paul Ramond directeur de formation, Monsieur Pascal Gelly directeur des études, et Corinne Couderc responsable apprentissage de l'administration de l'école et de l'organisme itii pour avoir répondu à mes questions et conseillé dans mes choix personnels.
- Monsieur Laurent Freund mon tuteur académique pour sa vision objective et les remarques constructives qu'il a apporté à mon projet.
- Toute l'équipe pédagogique de l'Ecole Nationale Supérieure des Mines de Saint-Etienne, et les intervenants professionnels de la formation électronique et informatique-industrielle qui ont assuré la partie théorique du diplôme et m'ont fait évoluer intellectuellement et personnellement.
- La promotion P17 qui ont toujours été présente pour moi.
- Mes proches qui m'ont soutenu pendant cette formation et ce travail de longue haleine.





La technologie RFID* est en plein essor, elle permet d'identifier les objets, d'en assurer la traçabilité ou bien d'en connaître les caractéristiques. Ceci se fait à distance grâce à une «étiquette» qui émet des ondes radios. La technologie RFID permet la lecture des étiquettes même sans ligne de vue directe et peut traverser de fines couches de matériaux (carton, boitier plastique, emballage, etc.).

Exemples d'applications opérationnelles :

- Dans le secteur de la logistique pour ce qui concerne la traçabilité.
- Dans les bibliothèques pour assurer la gestion des livres.
- Dans l'identification des animaux en remplacement du traditionnel tatouage.
- Elle fait son apparition dans le secteur de la téléphonie mobile pour des applications comme le paiement ou l'accès direct à des informations via internet.

Je travaille dans le secteur du semi-conducteur au sein de la société STMicroelectronics sur le site de Rousset et plus précisément dans le service qui est en charge de l'intégration des puces électroniques dans des applications finies.

L'arrivée d'un nouveau produit sur le marché passe par plusieurs étapes :

- Identification d'un besoin lié à une évolution technologique, à la demande d'un marché émergent ou bien au souhait d'un client recherchant un composant avec des fonctionnalités dédiées qu'il ne peut trouver dans le catalogue des produits standards.

- Définition du cahier des charges qui décrit les fonctionnalités et les caractéristiques du produit.

- Conception du produit par les ingénieurs design, dimensionnement, organisation et simulation du comportement des éléments qui vont composer le circuit.

- Routage des composants pour la fabrication des masques*.

- Fabrication du circuit.

- Vérification de la conformité du produit avec sa spécification, validations fonctionnelles (compréhension des commandes, réponses adaptées) et caractéristiques pour s'assurer de son domaine de fonctionnement dans son environnement (température, puissance du signal émis...).

- Si les résultats lors de l'étape précédente sont conformes, le produit sera lancé en production dans le cas contraire il retourne à l'étape de conception afin de corriger les problèmes identifiés.

Dans la suite du document je vais vous présenter le développement et la conception d'une application visant à la promotion d'une nouvelle puce électronique.

*RFID*: (de l'anglais) radio frequency identification, signifie Identification par radio fréquence.*

*masque* : modèle formé de zones opaques et transparentes, permet de définir le motif que l'on souhaite reproduire sur la plaquette.*





# Sommaire


Ecole Nationale
Supérieure des Mines
SAINT-ETIENNE

*RFID :* *De l’anglais «Radio Frequency Identification», signifie Identification par radio fréquence.*

masque : Modèle formé de zones opaques et transparentes, permet de définir le motif que l'on souhaite reproduire sur la plaquette.

RF : Radio Fréquence

*MMS :* *Microcontroller Memories & Smartcard*

*Wafer :* *Un wafer est un disque assez fin de matériau semi-conducteur (silicium). Il sert de support à la fabrication des puces électroniques.*

*Plaquette :* *Terme français pour désigner un wafer*.*

*EEPROM :* *De l’anglais «electrically erasable programmable read only memory».*

*Open-space :* *Espace de travail à plusieurs «ouvert».*

*Circuit RLC :* *Circuit linéaire intégrant une résistance R , une inductance L et une capacité C.*

*Datalogger :* *Système de filtrage et d'enregistrement de données dans un but de surveillance ou de statistiques, des données d'un système.*

*Hardware :* *Matériel informatique physique, par opposition au software, matériel logiciel.*

*PCB :* *De l’anglais «Printed-Circuit-Board» qui signifie «carte circuit imprimé»*

*Refactoring :* *Opération de maintenance du code informatique, peut se traduire par «remaniement».*

*Pinout :* *Noms et fonctions des contacts électriques d’une puce électronique.*

*IHM :* *Interface Homme Machine*

*One-Shot :* *Opération ponctuelle*

*driver :* *Pilote informatique*

*Firmware :* *Logiciel intégré à un matériel et permettant son exploitation.*

*Demonstration kit :* *Kit de démonstration, le produit est prêt à l’utilisation monté sur une carte. Livré avec un logiciel et les câbles utiles, il permet d’utiliser ses différentes fonctions très facilement.*

*Reference-Design* :* *C’est une application finie, intégrant le produit, il donne une idée de ce qui peut être réalisé avec le produit, il est livré avec les schéma de câblage le code source des logiciels.*

*Programmateur-debuggeur :* *Logiciel permettant de programmer et de suivre le déroulement d'un programme afin d'en repérer et corriger les dysfonctionnements*





# I. STMicroelectronics

## 1. L'entreprise STMicroelectronics

### 1.1. Histoire




*1.Figure : Historique de l'entreprise*

STMicroelectronics trouve ses origines en 1964 sous le nom de SESCO, entreprise qui fabriquait des composants discrets à Aix-en-Provence.

Puis en 1979 une alliance entre Saint-Gobain et National Semi-conducteur donne naissance à Eurotechnique, c'est à cette époque que l'on voit apparaitre le site de Rousset avec une usine de production 4 pouces.

En 1983 Thomson CSF reprend la société Eurotechnique pour devenir Thomson Semi-conducteur en 1985. Cette évolution permettra l'implantation d'une nouvelle unité de production 5 pouces. Rousset devient ainsi le siège de la division MOS.

En 1987 la société se nome désormais SGS-Thomson Microelectronics suite a la fusion avec l'entreprise italienne SGS-Microelectronica. En 1996 nouvelle évolution et l'unité de production 5 pouces est convertie en 6 pouces.

Enfin en 1998 après le retrait de Thomson c'est l'entreprise STMicroelectronics qui voit le jour avec en 2000 une nouvelle usine de production 8 pouces sur le même site de Rousset.

Aujourd'hui STMicroelectronics est coté à la bourse de New York, de Paris depuis 1994 et de Milan depuis 1998.

La société compte seize sites de production dans le monde, prés de 50000 employés et un chiffre d'affaire en 2007 de 10 Milliard de dollars (US).

Elle se place au cinquième rang mondial des constructeurs de semi-conducteur.

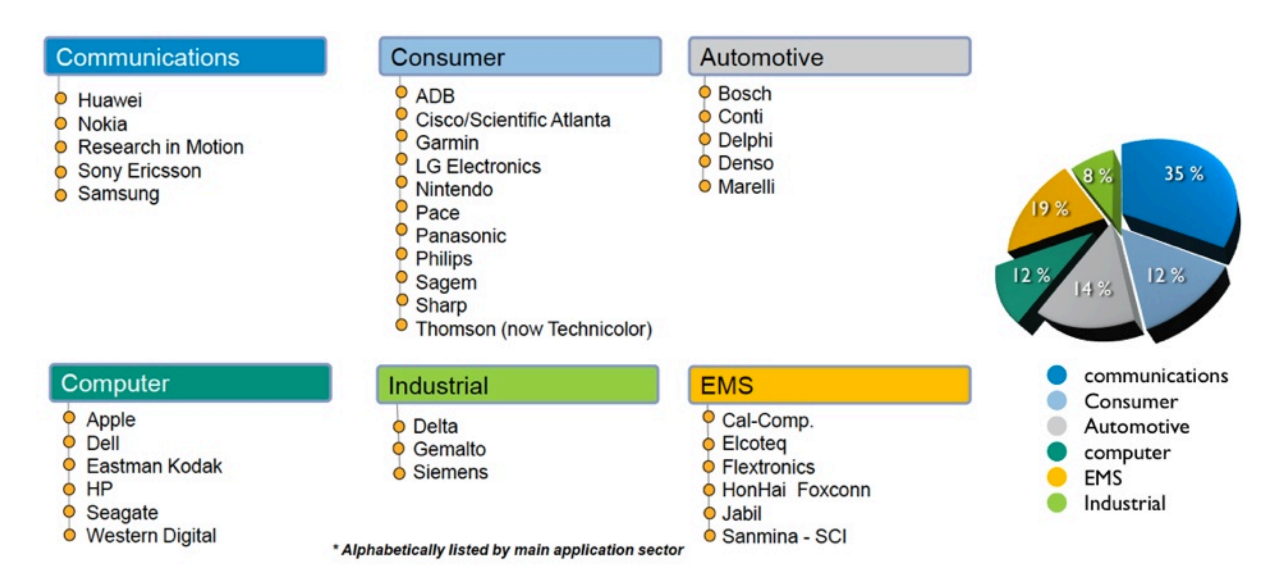



*2.figure : présence mondiale de STMicroelectronics*

## 1.2. Mission

La mission de STMicroelectronics est de satisfaire l'ensemble de ses partenaires, en étant un fournisseur de semi-conducteur profitable et disposant d'une large gamme de produits (figure 3). L'entreprise opère sur la majorité des segments de marché et a comme clients les plus grandes entreprises internationales.

Aujourd'hui STMicroelectronics vise 6 grands marchés stratégiques :

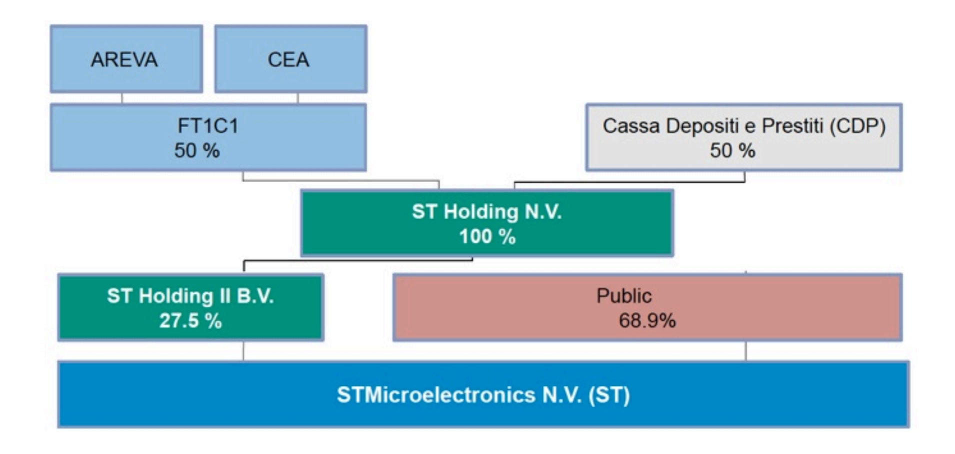



*3.figure : top 30 des clients STMicroelectronics (2009) et vente par segment de marché 2009*

## 1.3. Actionnariat

La valeur de STMicroelectronics est trop importante pour qu'une seule personne en soit propriétaire à 100%, même en recourant à l'endettement. Par ailleurs, la diversification des risque appelle à ne pas «placer tous ses oeufs dans le même panier.»

Afin de permettre à plusieurs personnes de devenir co-propriétaire de l'entreprise, le capital de STMicroelectronics est matérialisé par des actions. Le schéma ci-dessous indique la répartitions de celles-ci.




*4.figure : STMicroelectronics Actionnariat (au 31 décembre 2009).*

### 1.4. Organisation

L'unité RFID se situe dans le groupe MMS «Microcontroller Memories & Smartcard» encadré en rouge dans l'organigramme ci dessous.

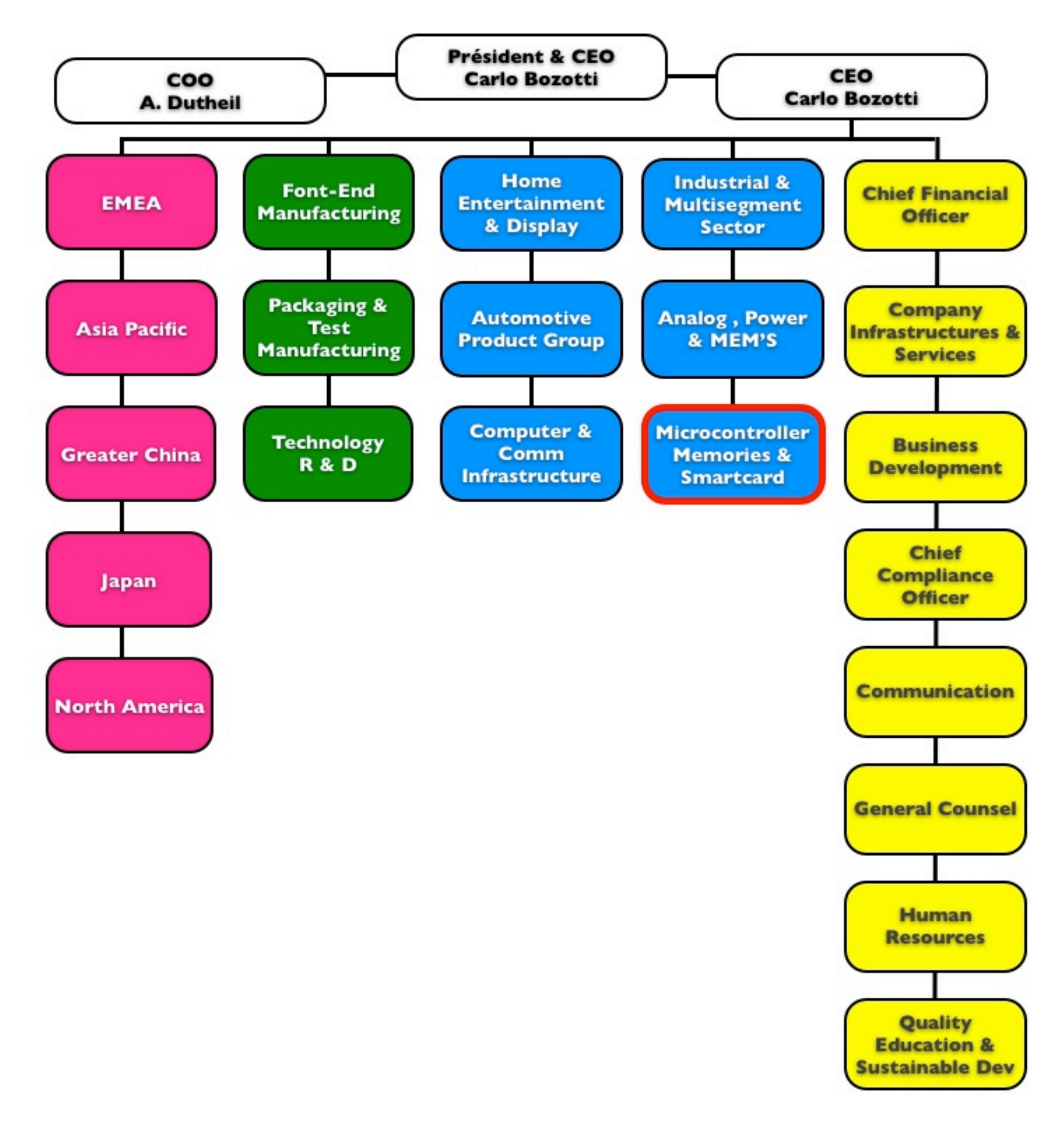
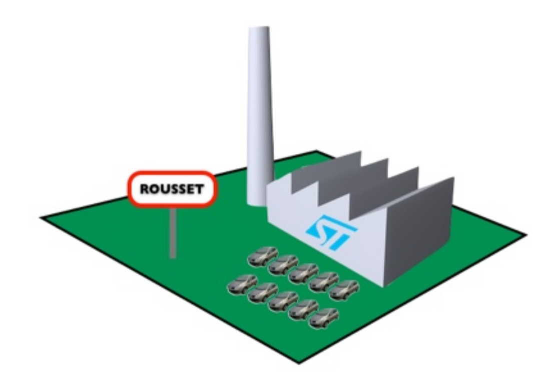



*Figure : Organigramme STMicroelectronics 2009*

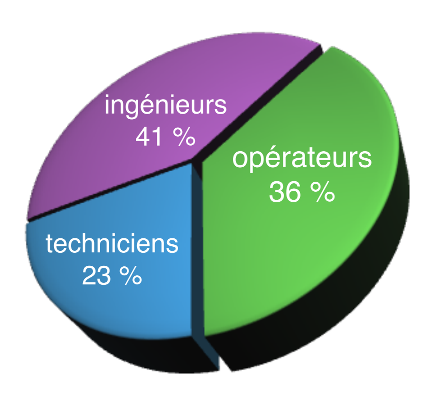
# 2. Le site de rousset

## 2.1. Caractéristiques




Le site est aujourd'hui équipé d'une usine de fabrication de technologie 8 pouces d'une capacité de 7000 plaquettes par semaine.
On y trouve également l'EWS (Electrical Wafer Sort) pour l'Europe qui a pour mission de tester les plaquettes avant la livraison chez le client.

Sont aussi présents sur le site des fonctions centrales telles que les ressources humaines France et STUniversity (branche de formation crée par l'entreprise).




On compte à peu près 3000 personnes sur le site de Rousset.
Trois catégories socio-professionnelles y sont représentées : les opérateurs, les techniciens et les ingénieurs.

*5.figure : Ordre de répartition des catégories sur le site de Rousset*

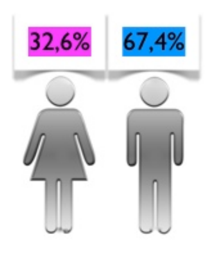
Ces 3000 employés travaillent :
- pour 60% d'entre eux en production
- pour 20 % en recherche et développement
- pour 20% dans l'administratif.










*6.figure : La répartition homme femme dans l'entreprise*

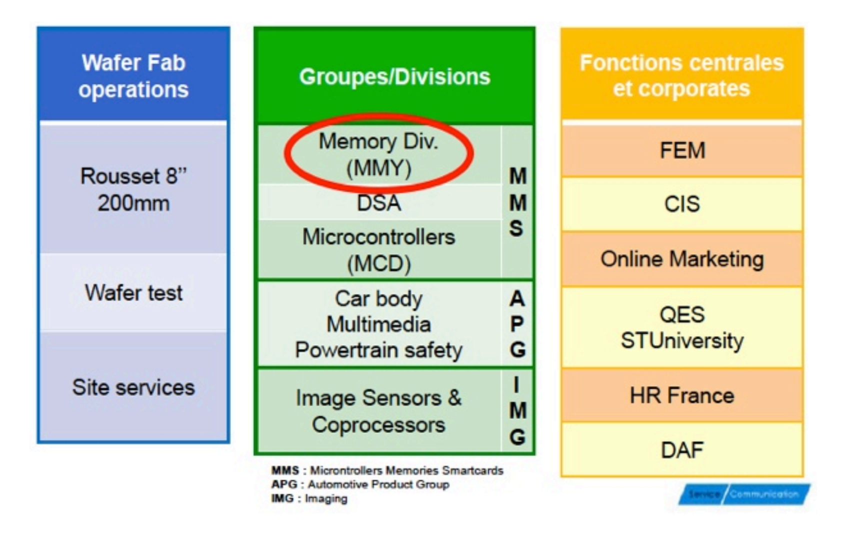
Au cours des dix dernières années, le site STMicroelectronics de Rousset a investi 1,6 Milliard d'euros, et consacre en moyenne 100 millions d'euros chaque année à la recherche et au développement.





## 2.2. Organisation

L'unité RFID se situe dans la division MMY «memories»




Trois divisions produits sont représentées sur Rousset :

- **APG** : Automotive Product Group
- **Imaging** : Capteur d'image CMOS pour toutes applications de type caméra, appareil photo et téléphone mobile
- **MMS** : Microcontroler Memory and SmartCard

Dans une division on trouve plusieurs entités qui ont chacune un rôle spécifique dans le développement et la vie d'un produit, dans la division MMY on peut par exemple retrouver :

- **Le marketing** : ce service est en contact direct avec le client, outre le fait de proposer des produits existants, il est aussi à son écoute et à celle du marché pour identifier les besoins du futur.
- **Les applications** : supportent le client, écrivent les spécifications des produits et explorent les utilisations possibles des produits.
- **Le service qualité** : veille à ce que les procédures de mise sur le marché soient respectées, analyse les défaillances des produits lors de leur développement et de leur production dans l'éventualité d'un problème.
- **Le planning** : Il gère les lancements en production des produits en fonction des commandes et la fabrication des lots engineering en phase de développement.
- **Opération & Manufacturing** : cette équipe est le lien entre la division et les diverses étapes de la production d'un produit.
- **Finance** : contrôle le budget alloué à la division.
- **Development** : conçoit d'un point de vue électronique les produits, et crée les masques nécessaires à la fabrication des wafer*.
- **Engineering** : Met en place les tests et caractérisations nécessaires pour garantir la concordance entre le produit et sa spécification.

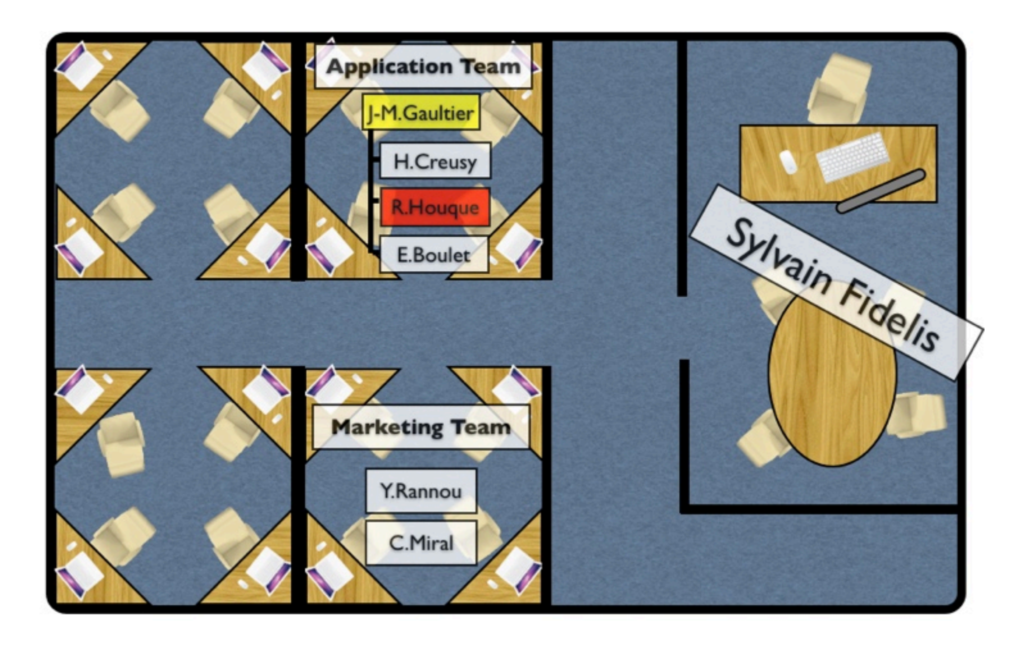
*wafer* : Un wafer est un disque assez fin de matériau semi-conducteur (silicium). Il sert de support à la fabrication des puces électroniques.*

# 3. L’ unité RFID

## 3.1. Organisation

Dans la division MMY sont développés des mémoires non volatiles de type EEPROM* et EPROM.

STMicroelectronics est numéro un mondial sur ce segment de marché. L’unité RFID est une des unités de la division EEPROM, elle réalise des mémoires RF* (sans fil). L’organisation du plateau est semblable au schéma ci-dessous.




*7.figure : représentation schématique de l’open-space* sur le plateau MMY*

Sylvain Fidélis encadre l’ensemble des équipes d’application et de marketing de l’unité RFID. J’occupe la place d’apprenti ingénieur aux cotés d'Hugues Creusy (ingénieur) et Emmanuel Boulet (technicien).
Nous sommes tous les trois sous le management de Jean-Marie Gaultier (mon tuteur) dans l’équipe d’application.

*EEPROM* : electrically erasable programmable read only memory*
*RF* : Radio fréquence.*

## 3.2. Portefeuille produit

### 3.2.1. Les mémoires (TAG)




L'unité RFID propose une gamme de produits mémoires RFID pour chaque protocole :
- ISO 14443 (Short Range) : partie haute du tableau ci-dessous
- ISO 15693 (Long Range) : partie haute du tableau ci-dessous

Ce sont des mémoires avec différentes capacités, réservées à différents domaines d'applications avec des niveaux de sécurité plus ou moins importants. STMicroelectronics vend ces mémoires sans antennes depuis maintenant deux ans. Les produits proposés sont répertoriés dans le tableau ci-dessous.

| Part number | Memory size | 64-bit unique ID | Anti-collision | Anti-clone | Package |
|---|---|---|---|---|---|
| ISO 14443 type B short-range/proximity | | | | | |
| SRi512 | 512-bit EEPROM | • | • | | Bumped dies, wafers |
| SRT512 | 512-bit EEPROM | • | • | | Bumped dies, wafers |
| SRiX4K | 4-Kbit EEPROM | • | • | • | Bumped dies, wafers |
| SRi4K | 4-Kbit EEPROM | • | • | | Bumped dies, wafers |
| ISO 15693 long-range/vicinity | | | | | |
| LRi64 | 64-bit WORM EEPROM | • | • | | Bumped dies, wafers |
| LRi2K | 2-Kbit EEPROM | • | • | | Bumped dies, wafers, VFN 2x3 |
| LRiS2K | 2-Kbit EEPROM | • | • | 3 passwords | Bumped dies, wafers |

### 3.2.2. Les lecteurs

L'unité RFID propose également des lecteurs RFID, cette fois uniquement pour le protocole : ISO14443 short range.



| Part number | Memory size | 64-bit unique ID | Anti-collision | Anti-clone | Package |
|---|---|---|---|---|---|
| ISO 14443 type B short-range/proximity | | | | | |
| CRX14 | | | | • | S016N |
| CR14 | | | | | S016N |

# 4. Mon poste

## 4.1. Ingénieur d'application

Le poste d'ingénieur d'application consiste à :

- Assister un client lorsqu'il intègre une de nos puces à son application, aussi bien lors de la conception que dans la vie de son application s'il rencontre un problème.
- Répondre aux questions des clients sur les fonctionnalités de nos produits, et la faisabilité d'applications via un outils mail appelé «support online».
- Rédiger les spécifications des produits et des applications.

Mon espace de travail : Je dispose d'un bureau situé dans un «open-space*», d'un ordinateur , d'une ligne téléphonique, d'une connexion internet et d'un accès au laboratoire RFID.

# II.Introduction à la RFID

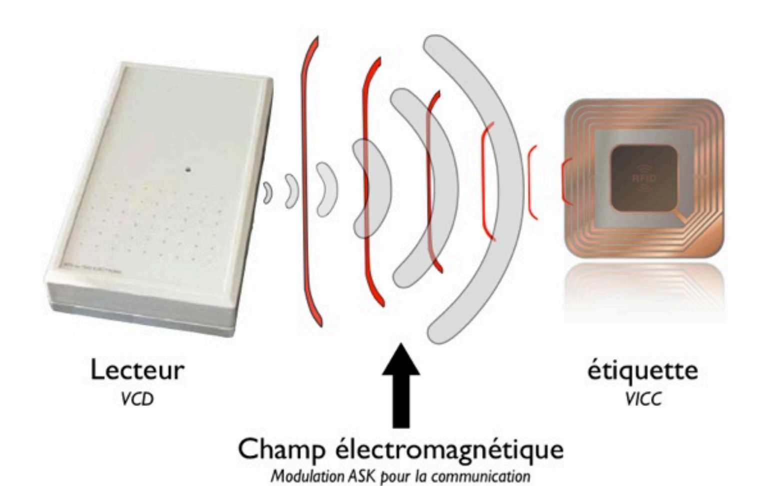
La RFID (de l'anglais : Radio Frequency IDentification) est un système d'identification par radio fréquence. La méthode consiste à enregistrer des informations dans une «étiquette intelligente».
La petite taille et la souplesse de ces étiquettes sans contact leurs permettent, d'être incorporées à des objets et même à des organismes vivants.

## 1. Description d'un système RFID

Un système RFID se compose de deux principaux éléments :

- **le lecteur** : c'est l'élément actif du système, il est composé d'une carte électronique et d'une antenne. Le lecteur peut être mobile ou fixe, il est souvent contrôlé par un ordinateur ou un système embarqué. Son antenne peut prendre différentes formes pour s'adapter à un maximum d'applications, on peut par exemple l'intégrer dans le cadre d'une porte.

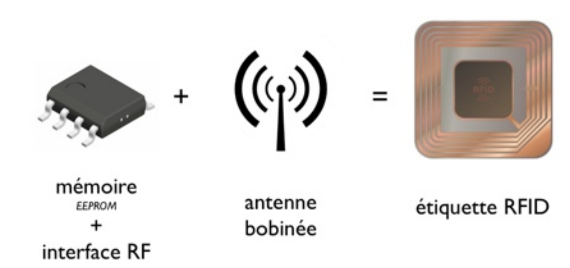
- **l'étiquette** : elle est totalement télé-alimentée par le lecteur, elle ne nécessite aucune source d'énergie. Elle est mise en fonctionnement dès qu'elle se trouve dans le champs électromagnétique émit par le lecteur. Notez qu'il existe des étiquettes actives, munies d'une batterie qui leur permet d'émettre un signal et donc de travailler à de plus grandes distances.




*VCD : Vicinity coupler device*
*VICC : VIcinity integrated circuit card*
*ASK : Amplitude Shift Keying*

## 2. Les produits RFID

Les produits RFID sont un assemblage d'une mémoire (*EEPROM** ) et d'une interface de communication RF* (qui répond à une norme ISO).
On connecte une antenne bobinée sur cette puce, et l'équipement est maintenant destiné à recevoir un signal sous forme d'ondes électromagnétiques et renvoyer immédiatement une réponse contenant une information pertinente.




Les produits RFID sont plus communément appelés «TAG» et on les retrouve le plus souvent sous forme d'étiquettes autocollantes. Elles peuvent être utilisées pour l'identification :

- d'objet : sur le même principe que le code barre.
- de personne : dans les nouveaux passeports (biométriques) et carte de transport.

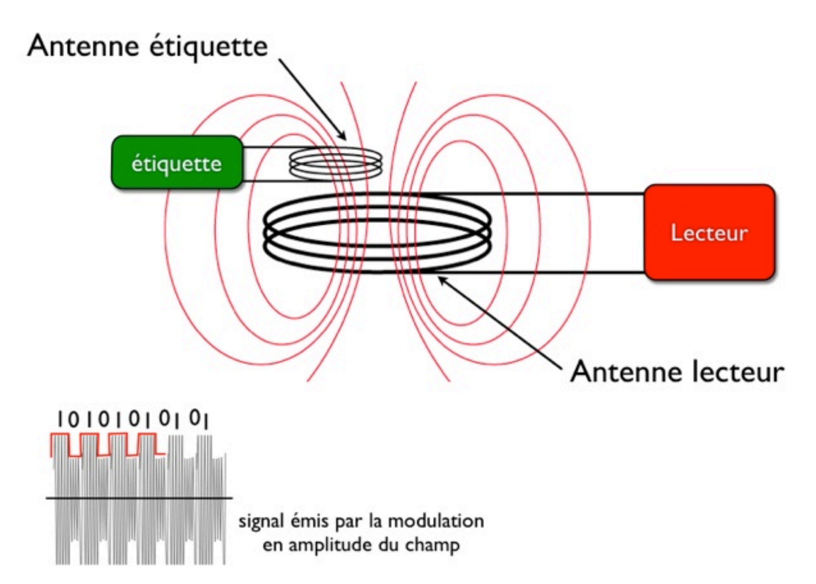




# 3. Principe de fonctionnement

Le lecteur active le ou les étiquette(s) qui baignent dans son champs électromagnétique, en leur fournissant l'énergie nécessaire.

Les deux éléments (étiquette et lecteur) se comportent comme des circuits accordés RLC*, l'optimum du transfert d'énergie dépend donc de la précision de la fréquence d'accord de ces deux circuits. Pour mieux comprendre, on peut assimiler le tag au secondaire d'un transformateur électrique.




Le lecteur peut transmettre de l'information en modulant en amplitude la porteuse à une fréquence particulière.

De son coté pour transmettre les informations qu'elle contient, l'étiquette fait varier sa charge pour moduler en amplitude le signal qu'elle reçoit. Cette modulation sera interprétée par le lecteur et transformée en code binaire.

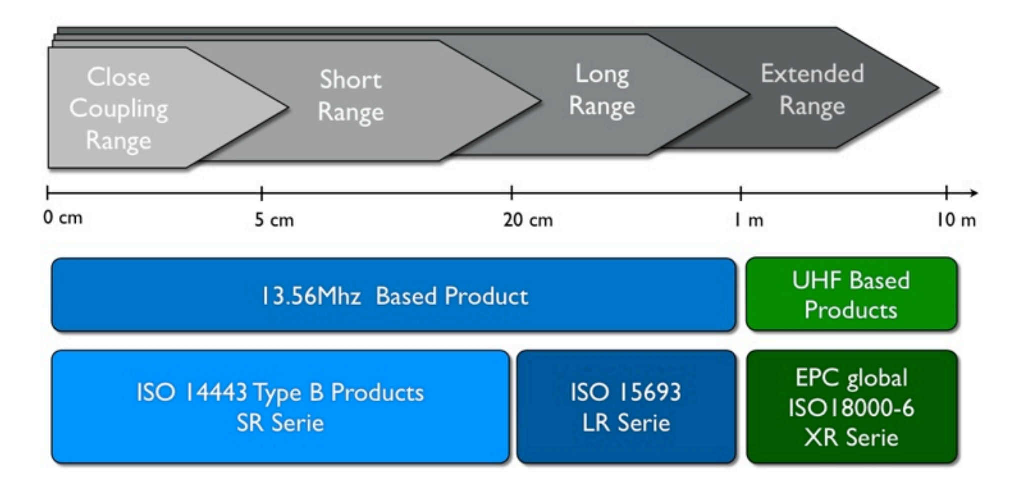
Le dialogue s'établit selon un protocole bien défini. Il en existe plusieurs en RFID dont les deux plus commun sont : ISO 14443 et 15693.

*8.figure : modélisation du couplage électromagnétique entre les antennes*

*RLC* : circuit linéaire intégrant une résistance R , une inductance L et une capacité C.*

# 4. Normalisation

Pour communiquer il existe plusieurs fréquences définies pour différents domaines d'applications.
En règle générale plus la fréquence est basse et moins le débit de données transmises est important.
Plus la fréquence est haute plus les distorsions et les difficultés pour traverser les milieux sont importantes.




*9.figure : synthèse des technologies et distances de fonctionnement*





# III.Le Projet

## 1. Introduction

Ce rapport détaille les différentes étapes de conception et réalisation d'un datalogger* de température. Celui-ci se présente sous forme d'étiquette intelligente utilisant le principe de la RFID* et de son interface logicielle permettant le contrôle via un PC. Cette application a pour but de promouvoir une nouvelle puce électronique nommée M24LR64.

*datalogger* : Système de filtrage et d'enregistrement de données dans un but de surveillance ou de statistiques, des données d'un système.*

## 2. L'application Datalogger

### 2.1. Description

Le datalogger est une application autonome fonctionnant sur batterie, elle permet d'enregistrer la température ambiante à intervalles réguliers.
L'utilisateur peut agir sur le système grâce à un lecteur RFID connecté à un PC pour démarrer/arrêter une acquisition, définir sa fréquence et télécharger les températures sauvegardées (figure ci-dessous).

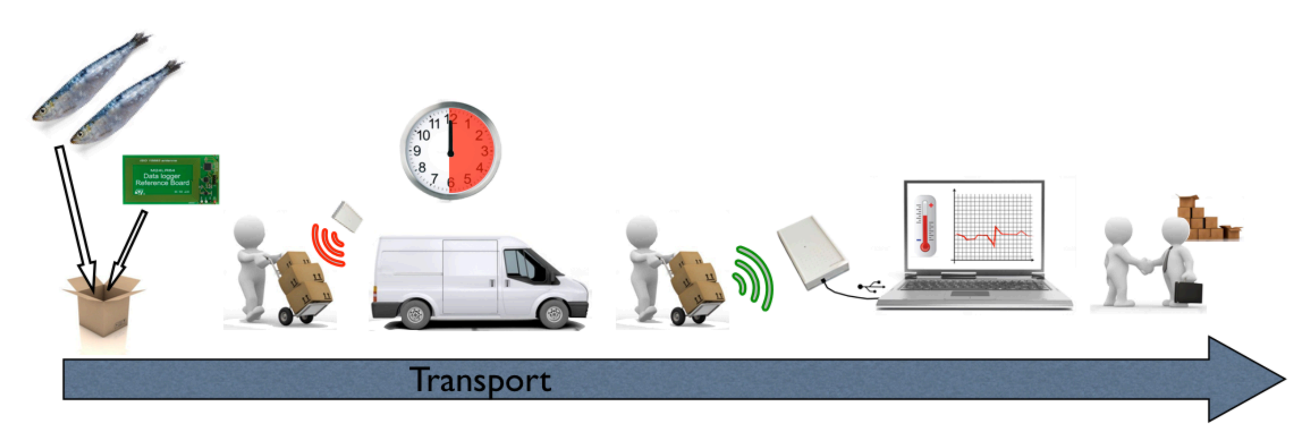



*10.figure : illustration du système complet.*

### 2.2. Exemple d'utilisation

Par exemple lors du transport de marchandises, l'application peut être utilisée dans un but de surveillance, ou de statistiques. Aujourd'hui, la chaine du froid concerne plus de la moitié des aliments consommés. La demande croissante en produits frais, génère des risques nouveaux, ces risques sont accentués par le phénomène de consommation de masse.

Prenons une situation concrète d'utilisation du datalogger: dans l'illustration ci-dessous, un livreur doit transporter du poisson.



**Description :** Le datalogger est placé avec la marchandise, puis activé grâce à l'association (logiciel PC + lecteur RFID). La livraison est effectué par un camion réfrigéré sur une durée de 6 heures. Durant le transport le datalogger enregistre la température à intervalles réguliers. La mise en oeuvre de ce dispositif, permet de vérifier les conditions de transport du produit ainsi, Le destinataire peut dès la réception et sans ouvrir les cartons afficher un graphique retraçant l'évolution de la température durant le trajet grâce au même matériel (logiciel PC + lecteur RFID). Il s'assure ainsi du respect de la chaine du froid pour la marchandise.

## 2.3. Objectif

L'objectif est de faire la promotion de la nouvelle puce électronique M24LR64 proposée par STMicroelectronics. Ce datalogger permet de mettre en avant la fonctionnalité du produit, et de le présenter au client sous la forme d'une application ludique. Il s'agit aussi de donner des idées au client afin qu'il développe ses propres applications autour du produit M24LR64.

En effet on peut imaginer tout un tas d'applications basées sur ce principe de datalogger. La carte électronique que j'ai développé intègre un capteur de température, ce qui en fait un datalogger de température, mais il peut devenir un datalogger de choc, de vibration, ou de lumière suivant le capteur choisi, et ce sans modifications majeures.

STMicroelectronics s'engage à accompagner le client souhaitant développer une application intégrant leur produit.

# 3. Le produit M24LR64

## 3.1. Origine

Comment est née la puce électronique M24LR64 ?

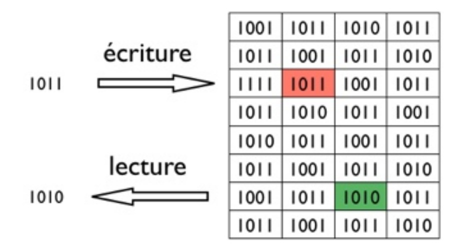
En tant que 1er fournisseur mondial de mémoires EEPROM réinscriptibles, la renommée de STMicroelectronics n'est plus à faire. Certains clients font donc appel à l'entreprise pour des demandes spécifiques. STMicroelectronics peut créer un composant «customisé» dédié à être vendu uniquement à ce client. Il faut bien entendu que la production reste rentable, cela implique une importante quantité de commande compte tenu du faible prix de vente de ce genre de mémoire (≈ 0.10 €/unit).

En développant un produit customisé pour un client particulier, STMicroelectronics acquiert un savoir faire et peut par la suite décider d'intégrer ce composant à son portefeuille produit. C'est de cette façon qu'est né le produit M24LR64.

## 3.2. Fonction

A quoi sert la puce électronique M24LR64 ?

Le produit M24LR64 est une mémoire, rappelons simplement que c'est un composant électronique très commun qui sert essentiellement à stocker des informations.




L'architecture d'une mémoire se traduit comme celle d'un tableau. Un système peut alors lire et écrire une valeur dans une cellule repérée par une adresse. (*voir le schéma ci-contre*)

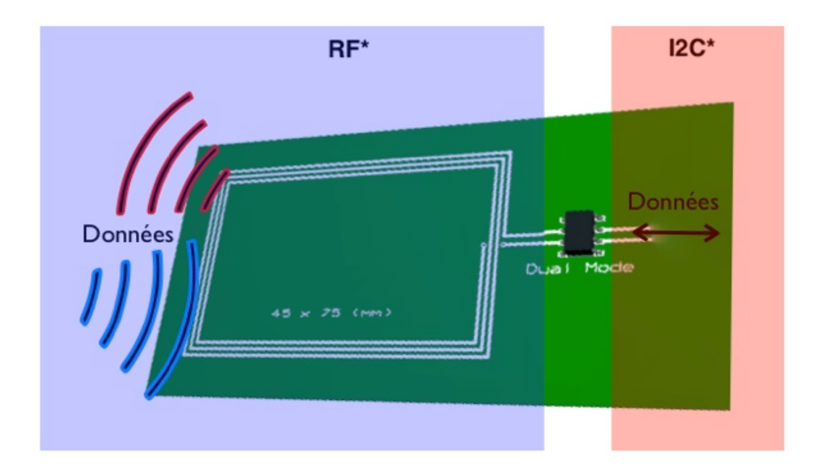
## 3.3. Innovation

Jusque là rien de révolutionnaire !

Mais elle se démarque des autres mémoires puisqu'elle est accessible par deux protocoles de communication :

- I2C* (de l'anglais : Inter Integrated Circuit Bus) «**avec contact**»
- RF* (de l'anglais : Radio Frequency) «**sans contact**»

Cela signifie en fait que cette mémoire, comme n'importe quelle autre mémoire standard, peut se trouver sur un circuit imprimé et communiquer grâce à des fils de connections.
Mais en ajoutant une antenne, il y aura aussi possibilité de communiquer avec cette mémoire à distance par ondes électromagnétiques «sans contact» (voir figure ci-dessous).




*11.figure : implantation de la puce sur une carte PCB*.*

Le fait de pouvoir accéder à la mémoire par deux protocoles (I2C & RF) fait du M24LR64 un nouveau concept de produit, le premier de la sorte pour STMicroelectronics.

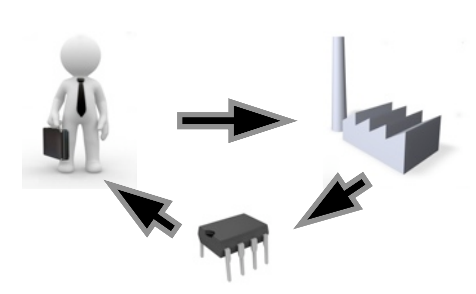
La mémoire double interface vise un large nombre d'application comme les équipements industriels ou médicaux, en effet son interface radio permet d'établir un contact avec un système évoluant dans un milieu hostile, ou se trouvant dans un endroit difficile d'accès.

Le fait de pouvoir envoyer des informations par ondes électromagnétiques à un composant qui est connecté au reste du système, permet de faire une mise à jour : Imaginons qu'il faille mettre à jour des imprimantes présentes dans le rayon d'un magasin. Aujourd'hui il faudrait déballer chaque imprimante et s'y connecter via USB ou autre, mettre l'imprimante sous tension, faire la mise à jour et remettre l'imprimante dans son emballage etc...

Si l'imprimante intègre une solution avec la mémoire double interface, il suffirait d'approcher un lecteur RFID des imprimantes concernées et d'effectuer leur mise à jour. Plus besoin de déballer, ou d'alimenter l'imprimante, la mémoire double interface ne nécessite que l'énergie fournie par le lecteur.

## 3.4. La commercialisation

- Le client achète le M24LR64 réalisé par STMicroelectronics.
- STMicroelectronics doit attirer de nouveaux clients pour continuer à se développer.
- Le M24LR64 reste la seule relation entre STMicroelectronics et le client.



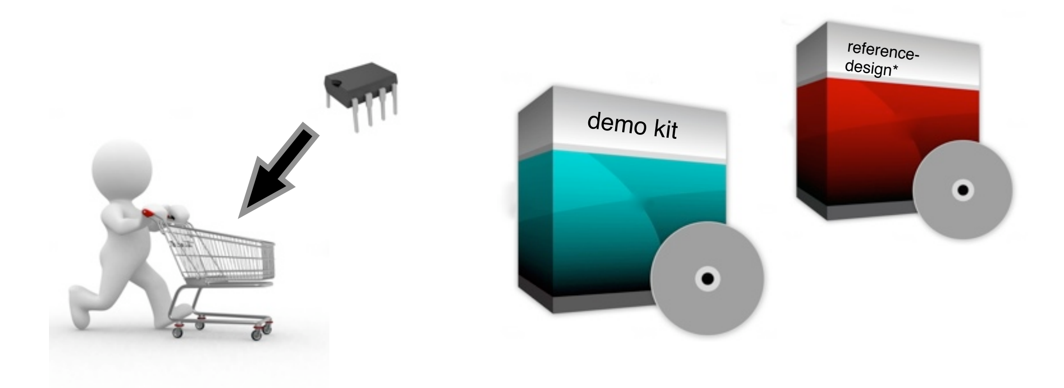
Le produit M24LR64 est imaginé et validé après l'expression du besoin d'un ou plusieurs clients, mais il va falloir promouvoir le M24LR64 pour attirer d'autres clients.

C'est à ce moment qu'une des techniques marketing de STMicroelectronics entre en jeu.
Pour faire connaitre le produit, STMicroelectronics mise sur deux méthodes :

- STMicroelectronics propose au client d'acheter un demonstration kit* : c'est une méthode d'accompagnement du client dans la découverte du produit, mettre à disposition les outils et répondre aux questions pour bien comprendre les capacités du produit.

- STMicroelectronics propose au client des reference-design* qui sont des applications toutes faites intégrant le M24LR64. Ils permettent de démontrer la faisabilité de certaines applications ou de donner des idées d'utilisations grâce à des exemples concrets. Ces références design sont livrées avec une documentation complète permettant aux clients de refaire leurs applications eux-même.

Au final, STMicroelectronics ne proposera pas seulement le M24LR64 aux clients qui en ont exprimé le besoin, mais vendra également le demonstration kit* et les reference-design* qui permettrons de toucher un public plus large.



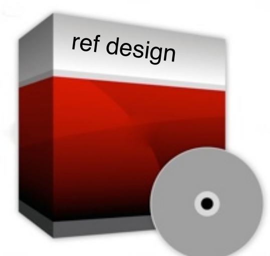

*demonstration kit *: kit de démonstration, le produit est prêt à l'utilisation monté sur une carte. Livré avec un logiciel et les câbles utiles, il permet d'utiliser ses différentes fonctions très facilement.*

*reference-design* : c'est une application finie, intégrant le produit , il donne une idée de ce qui peut être réalisé avec le produit, il est livré avec les schéma de câblage le code source des logiciels.*

# 4. Mon rôle / Ma mission

## 4.1. Description

Le datalogger de température que je réalise, est amené à être commercialisé sous la forme d'un reference-design*, avec son logiciel et sa documentation.



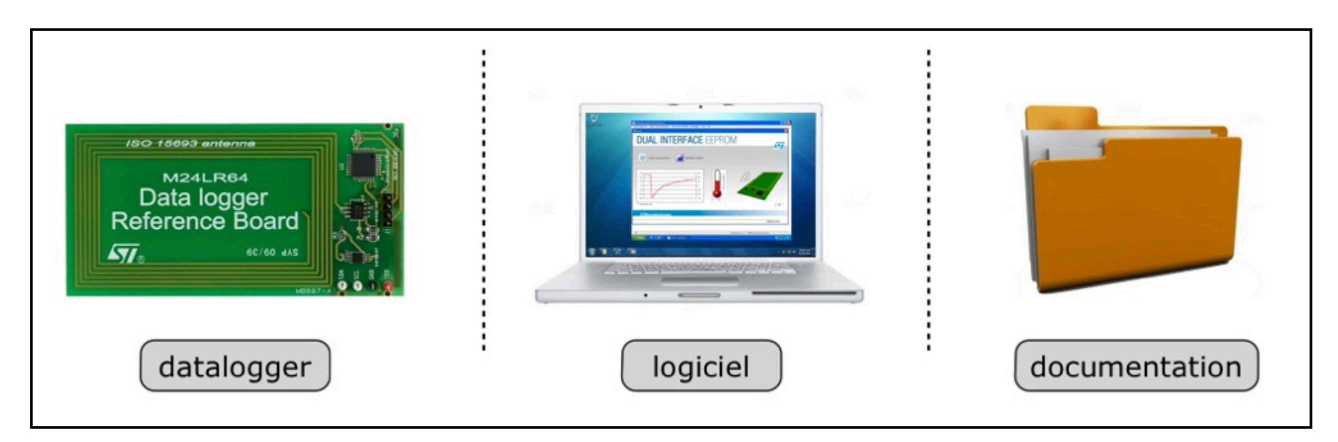

Cette application, doit aider les clients à développer leurs propres applications. Cette technique vise à la promotion d'un nouveau produit en démontrant ses capacités et la faisabilité d'un projet comme celui-ci à travers un exemple concret.

La réalisation de l'un des reference-design*, est un projet à part entière qui implique le management de ressources, la gestion des coûts et délais, la prise en compte des risques et la connaissance des techniques.

Le projet se déroule grâce à l'aide des personnes de l'équipe, un accès au laboratoire et au matériel mis à disposition, sur une période d'un an : s'étalant de Janvier 2009 à janvier 2010, dans les locaux de STMicroelectronics à Rousset.

**Le projet consiste à réaliser les trois parties ci-dessous :**

datalogger : étiquette intelligente qui mesure et enregistre la température.
logiciel : interface homme-machine pour contrôler le datalogger.
documentation : synthèse du développement et de la réalisation de l'application.




*12.figure : les trois grandes parties du projet*

# IV.Gestion de projet

## 1. Méthode

La gestion de projet pour mon application surtout orientée logiciel, puisqu'il consiste en la réalisation:

- d'un logiciel microcontrôleur
- d'un logiciel PC
- d'une carte électronique.

J'ai donc décidé de détailler la méthodologie utilisée pour la réalisation des parties logicielles.

### 1.1. Développement logiciel

Les méthodes de gestion de projet informatique connaissent au même titre que les technologies mises en oeuvre, une remise en cause permanente. La méthode utilisée pour le développement de ce projet est une des méthodes dites «Agiles».

Les méthode Agiles ont une démarche plus radicale que les méthodes classiques (Cycle en V etc...). De manière générale, leur but est d'augmenter le niveau de satisfaction du client tout en rendant le travail plus facile.

### 1.2. Méthodes agiles

Les fondements des méthodes agiles résident dans deux caractéristiques :

- Méthodes adaptatives plutôt que prédictives

Dans mon projet les exigences changent au cours du temps et le contexte évolue aussi (changement de hiérarchie directe). Les méthodes agiles se proposent de réserver un accueil favorable au changement. Elles adoptent une planification souple.

- Méthodes orientées vers les personnes plutôt que les processus.

Ces méthodes s'efforcent de travailler avec les spécificités de chacun plutôt que contre la nature de chacun. De cette façon mon projet se trouve être une activité plaisante où chacun se voit confier une part de responsabilité.

### 1.3. Crystal Clear

Il y a plusieurs types de méthodes agiles, suivant la taille de l'équipe et du projet. Celle qui est applicable à mon projet est la méthode Agile «Crystal Clear».

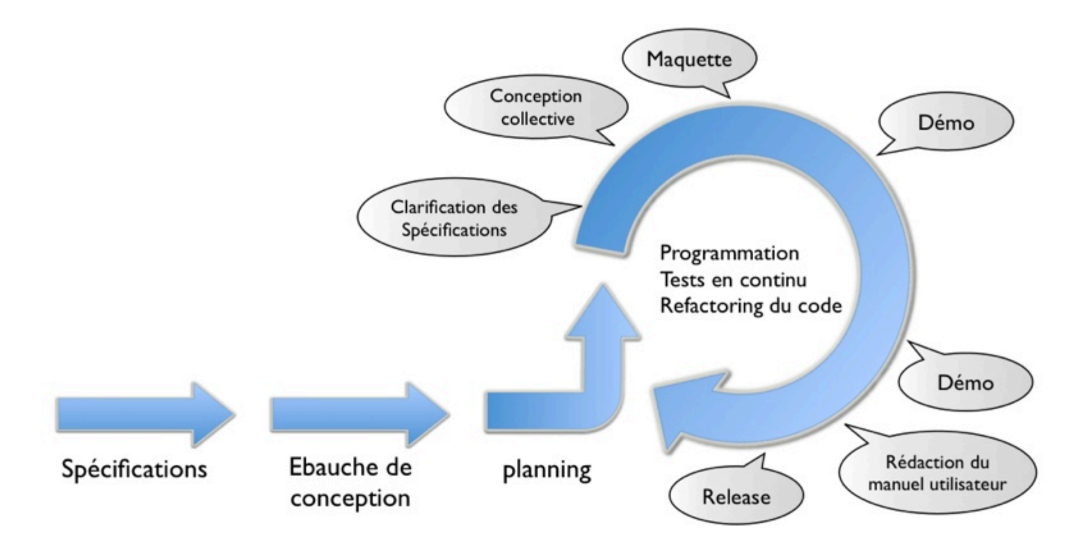
Crystal Clear est un cadre méthodologique très fortement adaptable aux spécificités de chaque projet. Ces méthodes ont été développées par Alistair Cockburn.

La communication est très importante et omniprésente dans ce type de méthode, l'équipe au sein de laquelle je développe l'application est composée des quatre personnes qui travaillent dans le même espace, c'est idéal pour une communication de proximité.

Dans un souci de souplesse face au changement, je dois livrer des versions de logiciel le plus souvent possible, ça permet au client d'avoir une partie utilisable de l'application sans attendre la fin du projet, et d'en faire une critique.

On soulignera la souplesse de la méthode qui permet aux personnes de travailler ensemble pour comparer leur code, se donner leurs avis et conseils. La méthode n'est pas directive sur la manière de coder non plus, puisque les personnalités de chacun sont prises en compte tant que le travail résultant est en accord avec les spécifications de départ.

### 1.4. Le cycle Crystal CLear




*13.figure : Cycle de vie Crystal Clear*

**Spécifications** : Une première phase consiste à interroger le client pour qu'il exprime son besoin. Une méthode assez répandue consiste à observer les futurs utilisateurs dans leur travail pour mieux comprendre leurs besoins et environnement de travail. En collaboration avec l'utilisateur un classement des fonctionnalités est établi pour savoir quelles sont celles à développer en premier.

- Le client direct est mon directeur marketing, il va être le premier utilisateur de l'application pour présenter le produits aux journalistes et à de potentiels futurs clients. J'ai donc établi un classement de ses attentes, avec un un ordre de priorité.

**Conception et Planning** : Puis il faut choisir les technologies qui seront utilisées pour la réalisation et une première ébauche pour donner une vision globale. Enfin juste avant d'entrer dans la phase itérative, il convient de planifier les itérations qui vont suivre.

- Je choisi les outils de développement en fonction de ceux utilisés et maitrisés par les ingénieurs de mon équipe, et je propose un croquis de l'interface logiciel et une date de première version livrable à mon client.

**Itération** : Il convient de mettre en place une maquette qui permettra de faire une démonstration aux utilisateurs, cette technique permet bien souvent de déceler des incompréhensions dans les besoins exprimés. Les test et le refactoring* du code sont omniprésents.

- J'organise une réunion pour présenter la maquette, et avoir un retour des personnes concernées, toujours dans l'optique de privilégier la communication et répondre au mieux aux attentes.

*refactoring* : opération de maintenance du code informatique peut se traduire par «remaniement».*

**Application des principes des méthodes agiles :**

Lorsque je livre une version du produit à mon client, je lui apporte directement et observe sa manière de procéder. Les méthodes agiles se basent énormément sur la communication et la coopération. Cela me permet de déceler des incompréhensions et de le modifier pour le rendre le plus intuitif et ludique possible.

J'ai invité mon tuteur académique Mr Laurent Freund à STMicroelectronics pour lui présenter le projet sur lequel je travaille. Comme s'il faisait partie intégrante de l'équipe, il m'a fait part de ses idées d'amélioration, les méthodes agiles réservent un accueil favorable au changement, j'ai donc pris en considération ses remarques et les ai intégré dans une nouvelle version.

Il y a eut une réorganisation hiérarchique dans laquelle j'ai changé de tuteur de manager direct et de manager supérieur. Cette réorganisation est intervenue pendant la période du projet, et a engendré une

nouvelle stratégie et un changement d'outil de développement et de langage de programmation. Les méthodes agiles préconisent un planning souple et une auto organisation des équipes. De cette manière mes collègues ont pu me former sur des outils et langages inconnus, pour repartir sur des bases solides.

### 1.5. La carte électronique

La méthodologie adoptée pour la réalisation de la carte électronique reste basée sur la méthodologie «Agile» pour ce qui concerne la communication mais l'approche est un peu différente en terme de changement.
La fonction que doit réaliser la carte datalogger est unique et figée dès le début on ne retrouve pas la souplesse du développement logiciel. Le prototype me permet néanmoins d'adopter la méthode «réalisation test en continu». Mais en terme de fonctionnalité il n'y a pas plusieurs versions, seul le design de la carte est amené à changer.

# 2. Budget

Mon manager a du recul sur ce genre de projet, avec son aide j'ai pu fixer un budget prévisionnel qui servira de référence pour suivre et contrôler les coûts en cours de réalisation du projet. Ce budget s'élabore à partir de l'organigramme des tâches et de la planification. En tant que chef de projet, ce budget est maintenant de ma responsabilité.

Pour chaque tâche définie (rubrique ci-dessous), je fais une estimation de la valeur des achats, de la sous-traitance et des coûts de main-d'oeuvre. Il faut ajouter à cette estimation budgétaire la rémunération d'un apprenti ingénieur sur une période d'un an.

Les achats :
- Composants nécessaires à la réalisation du prototype de la carte. (microcontrôleur, capteur de température, pile, connecteurs, etc.). Evaluation : 100€
- Outils hardware* de développement (programmateur-debugueur*). Evaluation 300€

La sous-traitance :
- Etude pour réalisation du PCB*. Evaluation 500€
- réalisation de «25» PCB*. Evaluation 25 x (30 à 50€) = 720 à 1250 €

Rémunération :
- Un ingénieur junior coûte environ 80 000€ à son employeur, dans l'entreprise je suis sous contrat d'apprentissage : entre les coûts de rémunération et de formation et les avantages fiscaux qui en découlent pour l'employeur, le calcul paraît profitable. Estimation <80 000 €.

Les coûts d'infrastructures :
Ils sont de l'ordre de deux cent cinquante euros par mois et par personne soit au total : 3000€.

L'estimation globale au commencement du projet pour une période d'un an du premier janvier 2009 au 31 décembre 2009 est de ≈ 85 000 €.

*Hardware* :* *Matériel informatique physique, par opposition au software, matériel logiciel.*
*PCB* :* *de l'anglais «Printed-Circuit-Board» qui signifie «carte circuit imprimé»*

*programmateur-debuggeur* : logiciel permettant programmer et de suivre le déroulement d'un programme afin d'en repérer et corriger les dysfonctionnements*

# 3. Planning

Le planning du projet présenté ci-dessous se décompose en plusieurs phases :

**Spécification et ébauche,** ces parties ont permis de donner une idée générale de l'application et de décrire son fonctionnement.

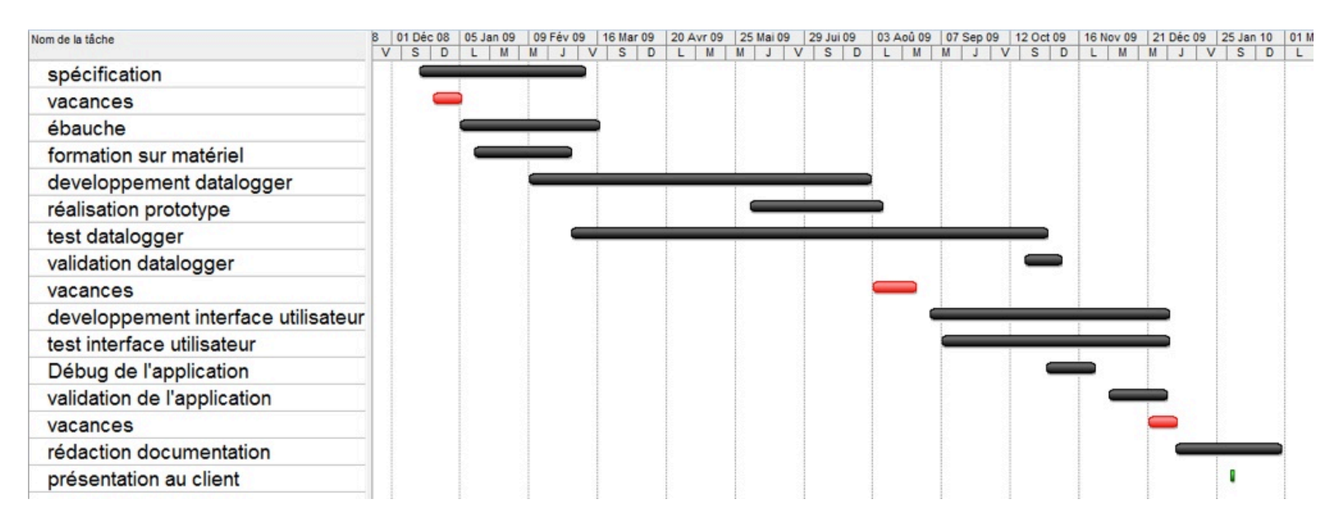
**formation sur matériel**, cette tâche est prévue dans le planning car le logiciel de développement que je vais utiliser pour programmer le microcontrôleur m'est inconnu.

**Réalisation et tests datalogger**, ces étapes sont prévues sur une période de temps assez longues, car il faut prendre en compte les temps d'étude et de livraison du sous-traitant avec lequel je vais travailler.

**Développement et tests de l'interface utilisateur,** Je n'ai pas besoin de formation pour les outils permettant de programmer cette IHM* car je les utilisais déjà lors de mon projet précédent. Je peux donc me faire une idée plus précise du temps à prévoir.

**Debug et validation de l'application,** Lorsque le datalogger sera validé je pourrai effectuer des test sur l'ensemble de l'application, ces étapes commencerons dés qu'il y aura une version fonctionnelle de l'interface utilisateur.

**Rédaction documentation,** dès que la première version de l'application complète sera validée, je rédigerai la documentation officielle STMicroelectronics.



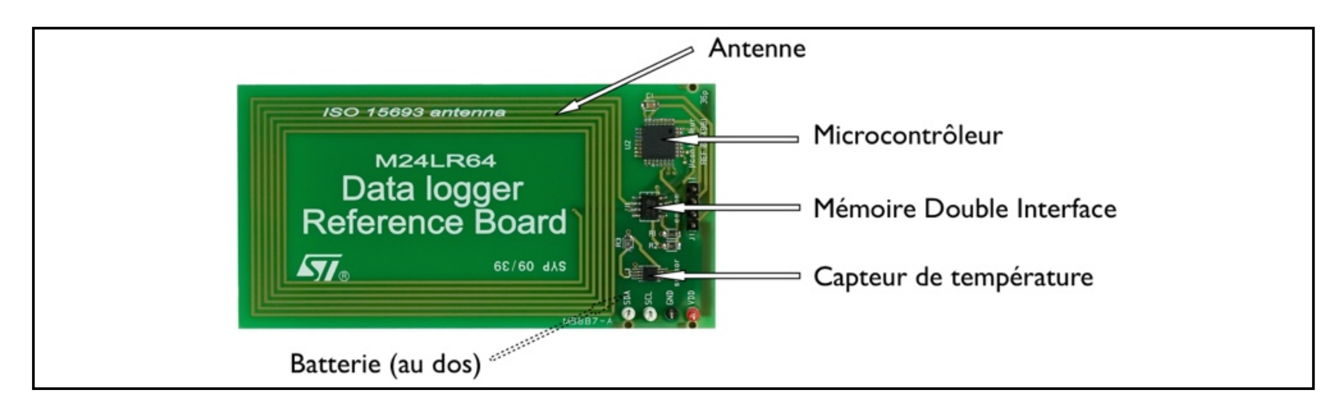

*14.figure : planning prévisionnel du projet*

J'ai alloué plus ou moins de temps aux tâches suivant mes niveaux de connaissance des outils et des techniques. Il faut prendre en considération le fait que la réalisation du projet en apprentissage, résume une semaine à 3 jours de travail effectif sur le projet industriel.

# V.Réalisation

## 1. Datalogger

J’ai entièrement réalisé la carte datalogger, elle est composée de trois puces électroniques, une batterie, et une antenne (voir figure ci-dessous).




*15.figure : composition du datalogger*

**L’antenne** : Elle permet au datalogger d’utiliser la technologie RFID pour se comporter comme un TAG et communiquer sans contact avec un lecteur.

- J’ai designé et accordé cette antenne à 13.56MHz, pour permettre un échange d’information avec une lecteur RFID basé sur la norme ISO 15693

**La mémoire double interface**: Elle permet d’enregistrer les informations du système et de les rendre accessibles par RF grâce à sa double interface.

- il m’a fallu l’intégrer sur le bus I2C et organiser son espace mémoire pour le fonctionnement de l’application afin d’établir un lien entre l’application et l’extérieur.

**Le capteur de température** : Il permet de faire l’acquisition de la température ambiante.

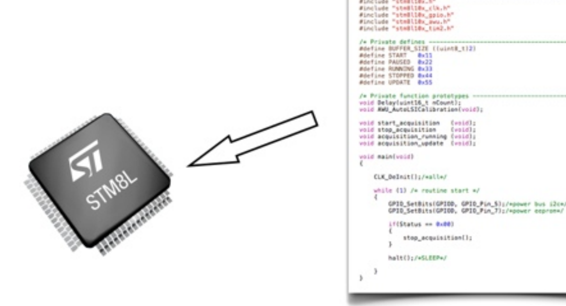
- j’ai choisi et sélectionné le composant, je l’ai configuré et intégré sur le bus I2C .

**La batterie** : Elle donne l'énergie nécessaire au système pour fonctionner en autonomie.

- J’ai réduit au maximum la consommation d’énergie du datalogger, et dimensionné la batterie nécessaire à une durée de vie correcte pour un reference-design*.

**Le microcontrôleur** : C’est une puce électronique dans laquelle l’utilisateur écrit programme (appelé Firmware*) pour lui faire réaliser les fonctions souhaitées. Avec ce programme le microcontrôleur devient «intelligent» puisqu’il peut réagir aux situations programmées.

- J’ai intégré cette puce sur le bus I2C et écrit le programme permettant d’enregistrer la température ambiante à intervalles réguliers.

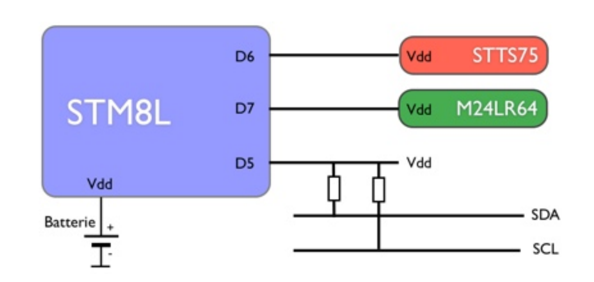



*Firmware* : logiciel intégré à un matériel et permettant son exploitation.*

### 1.1. Alimentation

L'alimentation électrique, est une fonction commune à toute la carte datalogger.
Habituellement, tous les composants nécessitant une alimentation sont reliés à la source d'énergie du système (alimentation secteur , pile, etc... ), mais dans ce cas, tous ces composants consomment de l'énergie en permanence.

L'application datalogger est amenée à fonctionner sur batterie. Je dois donc faire particulièrement attention à la consommation d'énergie.




La solution idéale serait de déconnecter les alimentations des composants lorsqu'ils ne sont pas utilisés dans le but de ne pas consommer de l'énergie inutilement. J'ai donc mis en place la solution ci-contre.

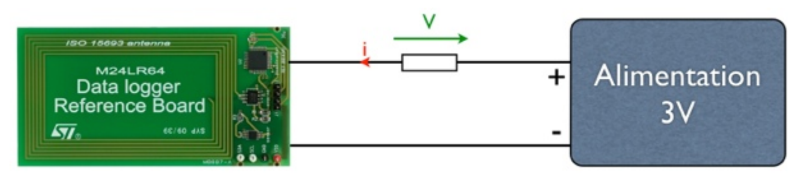
On voit que la batterie alimente uniquement le microcontrôleur et c'est celui-ci qui se charge d'alimenter ou non les autres composants et le bus I2C. De cette façon les alimentations sont gérées indépendamment les unes des autres.

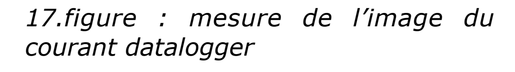
*16.figure : schéma gestion d'alimentation datalogger*

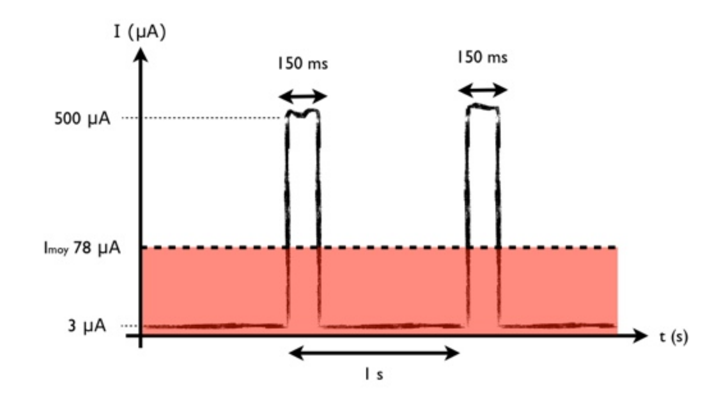
Avec cette solution, dans le cas ou je veux établir une communication uniquement entre le microcontrôleur et la mémoire, je peux décider de ne pas alimenter le capteur de température.

Le but a été de retrouver les courants de consommation annoncés dans les spécifications des produits.

J'ai validé cette solution en alimentant le montage avec une alimentation de laboratoire à travers une résistance en série. En plaçant une sonde différentielle aux bornes de cette résistance j'ai pu afficher sur un oscilloscope, l'image (V) du courant (i) consommé dans les différents cas de figure (écriture, lecture, prise température, etc...) ils sont expliqués dans la suite du document).





*17.figure : mesure de l'image du courant datalogger*




Je prends un exemple concret pour quantifier la durée de vie du système sur une batterie. Imaginons que le datalogger enregistre une température par seconde, en terme de consommation ça correspond à une moyenne de 78µA (voir schéma ci-contre).

*18.figure : courant moyen consommé par l'application en fonctionnement.*




J'ai opté pour une batterie type «pile bouton» qui délivre une tension de 3V et capable de fournir un courant de 255mA sur une heure (255mAh). Dans la configuration ci-dessus, cette pile pourrait alimenter le système pendant environ **3269 heures ≈ 4 mois et 15 jours**. Sachant que la mémoire serait de toute façon saturée bien avant, et le système passerait en mode «arrêt». Pour indication la pile peut alimenter le système en mode «arrêt» pendant un peu plus d'une année.

Le fait de consommer de l'énergie même en mode «arrêt» pose un problème car : si un des reference-design* reste stockée durant une année avant d'être livrée au client elle ne sera pas fonctionnelle à l'ouverture du colis (batterie faible). J'ai donc repris l'idée d'une grande majorité d'applications fonctionnant sur pile, à savoir le principe de la languette en plastique sur la pile faisant office d'isolant, et que l'utilisateur retire lors de la première utilisation. Avant d'opter finalement pour une solution avec un interrupteur ON/OFF qui permettra à l'utilisateur d'éteindre à volonté complètement le datalogger pour une économie de batterie optimale.

Avec cette gestion d'alimentation particulière, l'application datalogger peut être qualifiée d'application «Low Power» c'est un terme anglais qui désigne les applications à basse consommation d'énergie.

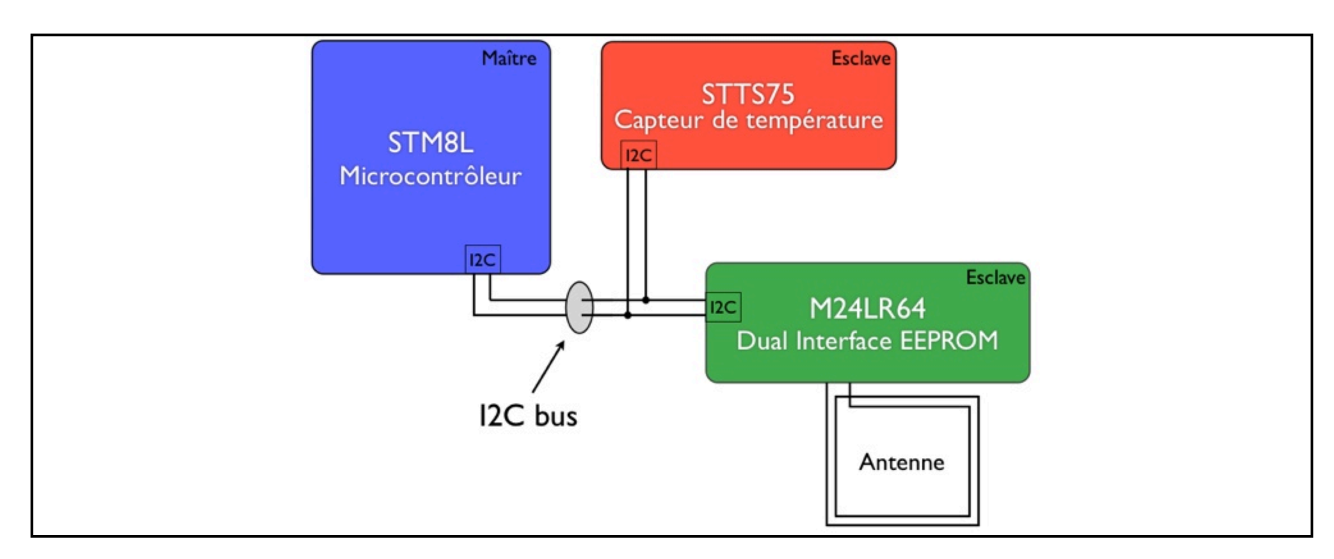
## 1.2. Interfaces de communication

Les trois composants de la carte datalogger doivent pouvoir échanger des données, et pour se faire ils utilisent leurs interface I2C.

Le bus I2C permet de faire communiquer entre eux des composants électroniques très divers grâce à seulement deux fils et une référence électrique:
- un signal de données (SDA)
- un signal d'horloge (SCL)
- une référence masse électrique (GND)

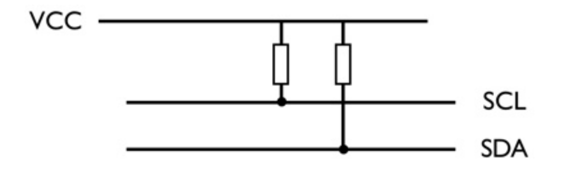
Je l'ai donc implémenté sur la carte datalogger pour y connecter le microcontrôleur en tant que «Maître*» , la mémoire double interface et le capteur de température en tant que «Esclaves*» (voir figure ci-dessous).

Maître & Esclave* : Les composants sont connectés au bus I2C soit en tant que «Maître» soit en tant que «Esclave». Les «Maîtres» donnent des ordres, les «Esclaves» se contentent d'y répondre




*19.figure : schéma de communication entre STM8L / M24LR64-R / STTS75*
*On remarquera dans la figure ci-dessous que la mémoire (M24LR64) est également connectée à une antenne, pour lui permettre de communiquer avec un lecteur RFID. Cette interface de communication est détaillée dans la partie [LOGICIEL].*

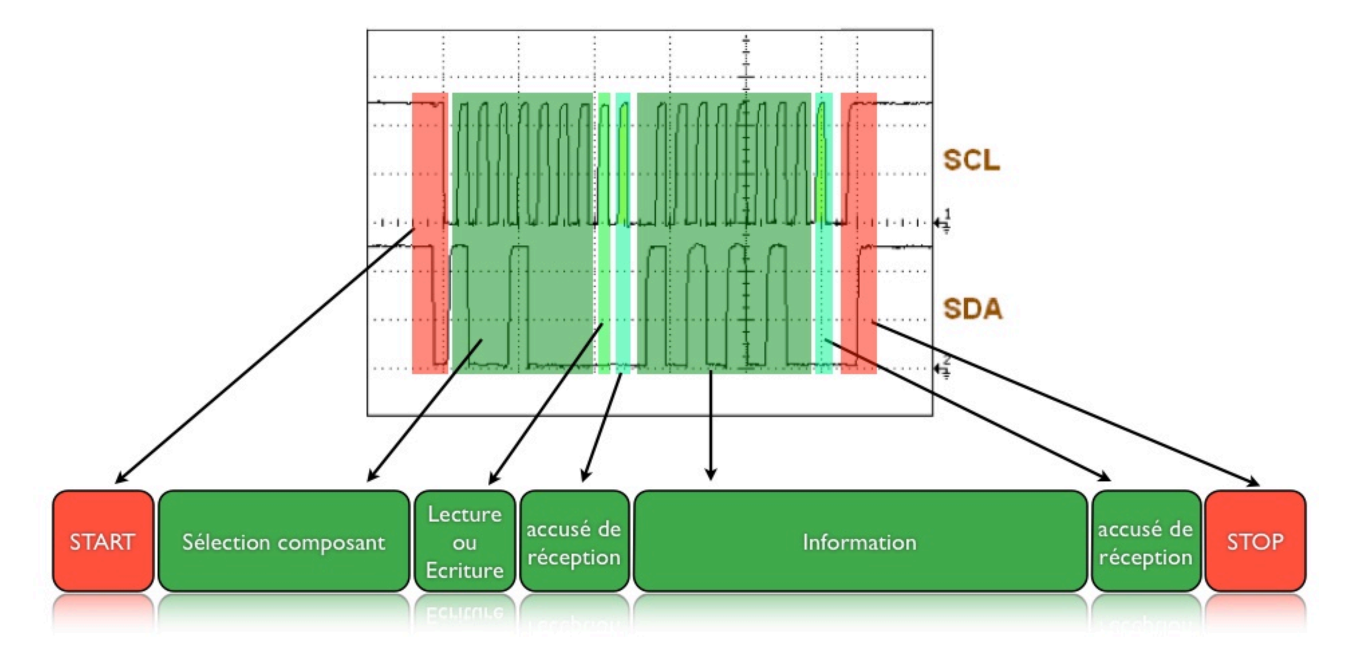
Les deux signaux du bus I2C doivent être alimentés par l'intermédiaire de deux résistances (appelées pull-up) sur la carte datalogger (voir schéma électrique ci-dessous)




J'ai dimensionné ces deux résistances toujours dans l'optique de consommer le moins possible d'énergie, tout en conservant la forme des signaux nécessaires au fonctionnement du bus I2C.

*20.figure : alimentation du bus I2C.*

Si l'on mesure les deux signaux sur un oscilloscope, ils ressemblent à la figure ci-dessous. La ligne SCL est une simple succession de créneaux à une fréquence de 100kHz à 400kHz qui permet de synchroniser la communication. Pour l'application datalogger je l'ai réglé à 100KHz toujours pour des souci de consommation d'énergie. La ligne SDA doit respecter un format bien spécifique que j'ai détaillé ci dessous. La validation du bus I2C se résume à vérifier la forme des signaux SCL et SDA sur un oscilloscope. La communication s'établie en respectant le format détaillé ci-dessous.




*21.figure : trame I2C standard*

- **[START & STOP]** Pour transmettre des données sur le bus I2C, il faut surveiller deux conditions particulières : la condition de départ et la condition d'arrêt.
- **[Sélection composant]** Le nombre de composants qu'il est possible de connecter sur un bus I2C est largement supérieur à trois, et le maître doit être capable de choisir quel esclave doit recevoir les données. Dans ce but, le maître envoi dans un premier temps l'adresse du composant.
- **[Lecture ou Ecriture]** Le maître indique ensuite s'il demande une lecture, ou s'il impose une écriture dans le composant concerné.
- **[accusé de réception]** Si l'esclave concerné répond les opérations continuent à se dérouler normalement. En revanche s'il n'y a pas d'accusé de réception de la part de l'esclave, le maître comprend qu'il y a une erreur et génère une condition d'arrêt.
- **[Information]** La zone information contient les données qui vont être écrite s'il s'agit d'une écriture. S'il s'agit au contraire d'une lecture, cette zone d'information va être renseignée avec les données voulues.

## 1.3. Les composants

Le choix des composants est une étape importante dans la conception d'application, il faut s'assurer qu'ils répondent à un certain nombre de critères :
Le type de boitier, la consommation, la vitesse, le bus de communication, le prix etc...
Il y a un critère supplémentaire pour ce genre d'application destinée à promouvoir un nouveau produit, c'est que les autres composants présent sur la carte doivent être des produits STMicroelectronics. Les trois composants choisis sont donc :

- la mémoire double interface : M24LR64-R
- le capteur de température : STTS75
- le microcontrôleur : STM8L101

Les trois parties suivantes expliquent : comment les choix ont été fait, comment ces composants fonctionnent dans l'application datalogger, les problèmes rencontrés, le genre de matériel annexe utilisé.




Mémoire double interface




Capteur de température




Microcontrôleur

### 1.3.1. Mémoire double interface [M24LR64]

**Rôle** : La mémoire comme son nom l'indique mémorise des données. Dans l'application datalogger, elle permet de stocker les températures acquises et les informations sur l'état du système, grâce à son interface radio, c'est elle qui permet de faire un lien avec l'extérieur.

**M24LR64** Mémoire EEPROM deux interfaces de communication (I2C & RF).



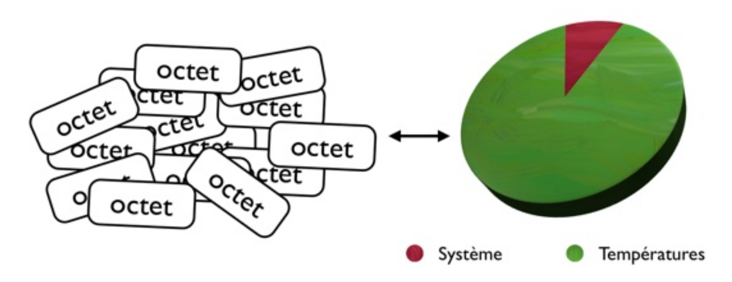

- Alimentation : de 1.8 à 5.5V
- compatible avec l' interface I2C
- compatible avec l'interface RF ISO 15693
- 64Kbit EEPROM organisé en :
  - 8192 octets en I2C
  - 2048 blocks de 32 bits en RF

Le M24LR64
est une mémoire de type EEPROM* accessible par deux protocoles de communication. Elle intègre une interface I2C qui nécessite une alimentation sur la pin Vcc. Elle intègre également une interface RF (ISO 15693) qui ne nécessite aucune alimentation puisque l'énergie est fournie par le champ électromagnétique du lecteur RFID.
(le digramme block et le pinout* de la mémoire double interface sont en annexe).

*pinout*: nom et fonction des contacts électriques de la puce*

#### 1.3.1.1. Organisation mémoire

Comme expliqué précédemment, la mémoire retient des informations de type binaire (1 ou 0). Celle-ci est organisée sous forme de lignes et de colonnes comme dans un tableau. Il suffit donc pour procéder à une lecture ou à une écriture d'indiquer l'adresse (le numéro de la ligne et de la colonne).




La mémoire MZ4LR64 met à disposition un espace de 8192 octets, je dois décider de l'organisation à appliquer à cet espace mémoire.

Je réserve un espace à la sauvegarde des températures (schéma ci-contre en vert) et un espace pour stocker des informations sur le système (schéma ci-contre en rouge).

Il m'a fallu trouver un compromis entre :

- enregistrer un nombre maximum de températures.
- Avoir suffisamment d'informations sur l'état et la configuration du système.

J'ai finalement organisé cet espace mémoire comme dans le tableau ci-dessous.

| numéro secteur | RF block adresse | i2c octet adresse | bit [31:24] | bit [23:16] | bit [15:8] | bit [7:0] |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | RFU | Delay | Overwrite | Status |
| 0 | 1 | 4 | RFU | RFU | Nb Temp[1] | Nb Temp[0] |
| 0 | 2 | 8 | Temp2 [1] | Temp2 [0] | Temp1 [1] | Temp1 [0] |
| 0 | 3 | 12 | Temp4 [1] | Temp4 [0] | Temp3 [1] | Temp3 [0] |
| 0 | 4 | 16 | Temp6 [1] | Temp6 [0] | Temp5 [1] | Temp5 [0] |
| 0 | 5 | 20 | Temp8 [1] | Temp8 [0] | Temp7 [1] | Temp7 [0] |
| ... | ... | ... | ... | ... | ... | ... |
| 63 | 2047 | 8188 | Temp4092 [1] | Temp4092 [0] | Temp4091 [1] | Temp4091 [0] |

*On remarquera que les adresse pour accéder en I2C et en RF sont différentes, ce qui explique les deux colonnes adresse dans le tableau ci-dessus.*

*22.figure : organisation mémoire du M24LR64*

Dans ce tableau on retrouve :

- (en rouge), une partie qui sauvegarde des informations propres au fonctionnement système comme (l'espace mémoire disponible, la fréquence d'acquisition, etc.). Elle occupe 8 octets sur un total de 8192, soit moins de 1/1000 de l'espace mémoire.
- (en vert), l'autre partie est réservée à la sauvegarde les températures mesurées. 4086 valeurs peuvent être enregistrées dans la mémoire.

Les 8 octets d'information système (en rouge), vont me permettre de connaître et d'utiliser toutes les informations nécessaire au fonctionnement de l'application. Ils sont détaillé ci-dessous.

**Status** :

| bit [31:24] | bit [23:16] | bit [15:8] | bit [7:0] |
|---|---|---|---|
| RFU | Delay | Overwrite | Status |
| RFU | RFU | Nb Temp[1] | Nb Temp[0] |

J'ai mi en place un octet «status» qui pourra indiquer à tout moment l'état du système. Il me suffira de lire cet emplacement de la mémoire pour savoir si le datalogger est à l'arrêt ou en fonctionnement.
Dans un soucis de flexibilité je défini cinq états différents :
**Start - Paused - Running - Stopped - Update**

**Overwrite** :

| bit [31:24] | bit [23:16] | bit [15:8] | bit [7:0] |
|---|---|---|---|
| RFU | Delay | Overwrite | Status |
| RFU | RFU | Nb Temp[1] | Nb Temp[0] |

Je décide également de mettre en place un octet appelé «Overwrite», car en effet lorsque l’application enregistre des températures dans la mémoire, au bout d’un certain temps celle-ci arrive à saturation. A ce moment là deux options sont possibles :

- Continuer à enregistrer de nouvelles températures en écrasant les anciennes.
- S’arrêter et ne plus enregistrer de températures supplémentaires.

L'utilisateur choisira une de ces deux options en écrivant la valeur **Authorized** ou **Non_Authorized** à cet emplacement mémoire.

**Delay** :

| bit [31:24] | bit [23:16] | bit [15:8] | bit [7:0] |
|---|---|---|---|
| RFU | Delay | Overwrite | Status |
| RFU | RFU | Nb Temp[1] | Nb Temp[0] |

Si l’octet «Overwrite» ci-dessus est une option qui se révèle facultative, L’octet «Delay» est primordial. C’est à cet emplacement mémoire que je stock la fréquence d’acquisition. L’utilisateur y écrit au choix une des valeurs proposées :
**256ms** - **512ms** - **1s** - **2s** - **12s** - **30s** .

Imaginons que l’octet «Delay» contienne la valeur (2s), cela signifie que le système mesure et sauvegarde une température toutes les deux secondes.

**Nb_Temp** :

| bit [31:24] | bit [23:16] | bit [15:8] | bit [7:0] |
|---|---|---|---|
| RFU | Delay | Overwrite | Status |
| RFU | RFU | Nb Temp[1] | Nb Temp[0] |

Il me faut connaître la place disponible dans la mémoire à chaque instant, je décide donc de stocker le nombre de température acquise. Ce nombre pouvant aller jusqu'à 4092 je n’ai pas d’autre choix que d’utiliser deux octets, en effet un seul octet ne me permet de coder que 256 valeurs différentes, contre 65536 pour deux octets.

C’est la seule information que j’ai décidé de stocker sur le nombre d’acquisition, puisqu’à partir de ce nombre je peux déduire la place disponible dans la mémoire et connaître en temps réel le nombre de températures enregistrées.

**RFU** :

| bit [31:24] | bit [23:16] | bit [15:8] | bit [7:0] |
|---|---|---|---|
| RFU | Delay | Overwrite | Status |
| RFU | RFU | Nb Temp[1] | Nb Temp[0] |

je garde une marge de manœuvre dans le cas où il me manquerais des informations pour d’éventuelles évolutions de l’application.
Dans cet optique je note ces octet inutilisés «RFU» c’est un terme anglais signifiant (Reserved for Future Use) ils n’ont aucune utilité aujourd’hui, mais ils sont réservés pour un usage futur.

**Temp** :

| | | | |
|---|---|---|---|
| Temp2 [1] | Temp2 [0] | Temp1 [1] | Temp1 [0] |
| Temp4 [1] | Temp4 [0] | Temp3 [1] | Temp3 [0] |

Le capteur de température délivre des valeurs codées sur deux octets, je les enregistres aux format brut dans la mémoire pour éviter au microcontrôleur de faire des calculs, donc je réserve par deux les octets nommés Nb_Temp .
*(Le format de température est expliqué dans la partie annexe).*





Dans ce schéma j'illustre la correspondance entre les états détaillés ci-dessus, et les action qui permettent de passer d'un état à l'autre. A la mise sous tension (ON/OFF pile bouton) le système s'initialise tout seul en état «STOP».

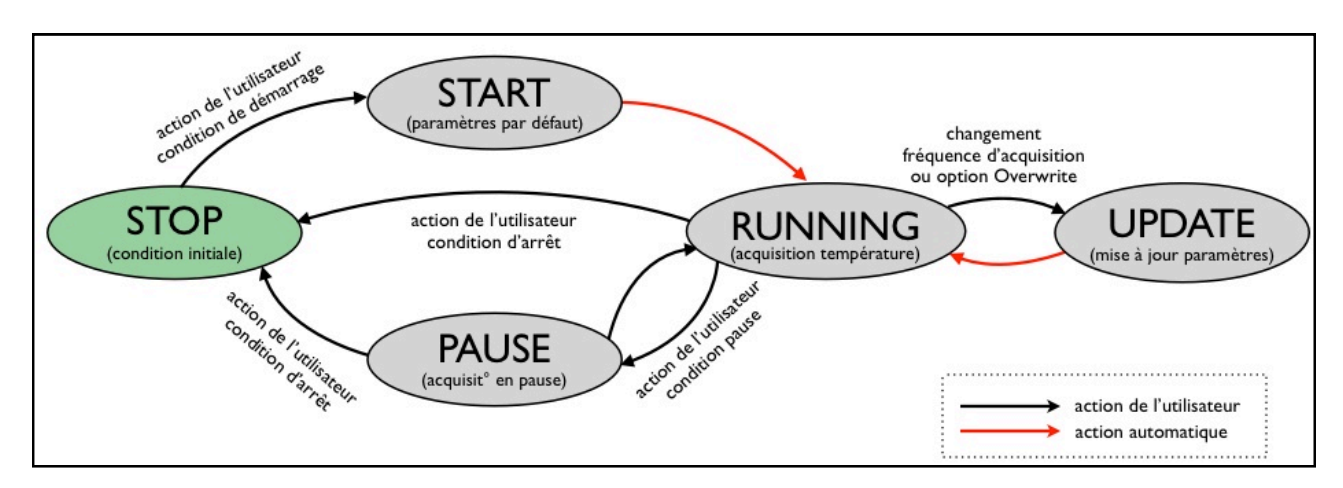



*figure : diagramme de correspondance des états du système*

### 1.3.1.2. Les commandes I2C

Rappelons que le bus I2C permet aux composants de communiquer entre eux et pour être identifié par les autres composants sur ce bus I2C, le M24LR64 a un identifiant (adresse) , celui-ci est : «1010 0000» en valeur binaire (0xA0 en valeur hexadécimale).
Les commandes I2C sont envoyées par le «Maître», pour l' «esclave» , La mémoire double interface M24LR64 est connectée sur le bus en tant qu' «esclave», c'est-à-dire qu'elle reçoit des ordres auxquels elle doit répondre.

J'ai donc construit les trames I2C que le microcontrôleur STM8L (Maître) va envoyer à la mémoire M24LR64 (esclave).

Je dois réaliser deux types d'opérations avec la mémoire M24LR64 :
- Ecriture
- Lecture

**Exemple d'une écriture de la mémoire M24LR64** :




*23.figure : commande I2C écriture d'un octet.*

J'ai reconstitué cette trame à l'aide de la bibliothèque i2c du microcontrôleur «stm8l10x_i2c.c». Dans cette bibliothèque, on trouve des fonctions comme :

```
• void I2C_GenerateSTART (FunctionalState NewState)
• void I2C_GenerateSTOP (FunctionalState NewState)
• uint8_t I2C_ReceiveData (void)
• void I2C_Send7bitAddress (uint8_t Address, I2C_Direction_TypeDef I2C_Direction)
• void I2C_SendData (uint8_t Data)
• etc.
```

En utilisant celles-ci je peux créer mes propres fonctions «générique» comme par exemple : une fonction : « ecrire_memoire » dans laquelle il me suffira de renseigner l'adresse à laquelle je veux accéder et la donnée que je veux y écrire. Le code de cette fonction est disponible en annexe dans la partie «i2c_ee.c» je l'ai crée sous le nom de :

```
void I2C_EE_PageWrite(uint8_t* pBuffer, uint16_t WriteAddr, uint8_t NumByteToWrite)
```

pBuffer : est l'information à écrire
Writeaddr : est l'adresse à laquelle écrire l'information

NumByteToWrite : est la longueur de l'information à écrire (nombre d'octet)

J'utilise cette fonction dés que j'ai besoin d'écrire des informations ou des températures dans la mémoire.
**Exemple d'une lecture de la mémoire M24LR64 :**




*24.figure : commande I2C lecture d'un octet*

Dans la trame de la lecture on discerne bien les conditions START/STOP ici un peu particulières puisqu'une lecture, est en fait la succession d'une écriture puis d'une lecture.

Comme pour l'écriture, j'ai reconstitué une fonction «générique» :

```
void I2C_EE_BufferRead(uint8_t* pBuffer, uint16_t ReadAddr, uint8_t NumByteToRead)
```

pBuffer : est l'information lue
Writeaddr : est l'adresse à laquelle lire l'information
NumByteToWrite : est la longueur de l'information à lire (nombre d'octet)

#### 1.3.1.3. Difficultés rencontrées

Au commencement du projet datalogger, la mémoire M24LR64 n'était qu'au stade de prototype. Cette mémoire double interface a été imaginée par des ingénieurs puis conçue par des équipes de designers. La première version de la puce est sortie de l'usine de Rousset début 2009 en seulement quelques exemplaires.

Les premières difficultés que j'ai rencontré dans la réalisation de ce projet, furent :
- de me procurer quelques échantillons du produit
- que ces échantillons soient fonctionnels.

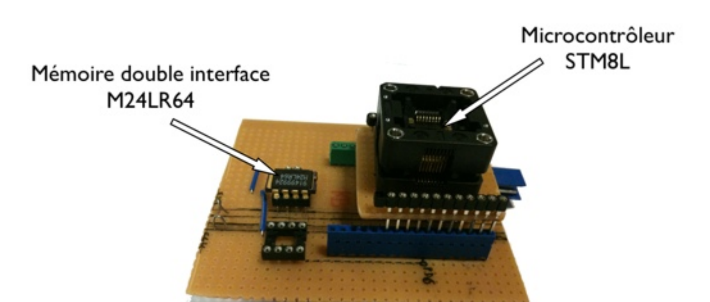
Car sur les premières versions des puces, le rendement d'un wafer* n'est pas maximum. En effet, même si la technologie est maitrisée, on peut s'attendre à un pourcentage de puces «bonnes» approchant les 70% ces résultats s'amélioreront de version en version.

Une fois les quelques échantillons à disposition, je peux commencer à tester les fonctionnalités du produit que je compte utiliser dans l'application datalogger.
Mon rôle est aussi dans ce cas de faire remonter les erreurs ou les comportements anormaux de la puce que je pourrais remarquer. Cela permet d'établir une liste de points à rectifier sur la prochaine version.

#### 1.3.1.4. Validation

Le projet débute, alors que l'entreprise STMicroelectronics traverse une période très compliquée financièrement, puisque la crise économique d'automne 2008 touche sévèrement le marché du semi-conducteur. Dans une optique d'économie, certains services de STMicroelectronics verront leur activité totalement arrêtée, ce n'est pas le cas dans mon service, mais les commandes ou les achats de matériel ont été suspendus.

La validation de ce composant dans l'application est donc faite sur un prototype que j'ai entièrement réalisé, pour faire face au gel des commandes et achats (voir photo ci-dessous).




Grâce à cette carte, j'ai pu valider l'alimentation de la mémoire (M24LR64) via une pin du microcontrôleur (STM8L) et la communication I2C entre le microcontrôleur entre ces deux composants : écriture et lecture de la mémoire.

*25.figure : prototype pour la validation communication I2C*

### 1.3.2. Capteur de température [STTS75]

**Rôle** : Il mesure la température ambiante et la convertie en format numérique.

**STTS75** Capteur de température numérique de haute précision.

- Alimentation : de 2.7 à 5.5V
- faible consommation de courant : 75µA à 3.3V
- compatible avec l' interface I2C
- gamme de mesure : -55°C à 125°C
- précision : ± 0.5°C dans la gamme -25°C à 100°C.
- mode de fonctionnement basse consommation d'énergie
- encodage de la température programmable de 9 à 12 bits

Le STTS75 est un capteur de température de haute précision. Il intègre un convertisseur Analogique-numérique qui lui permet de délivrer une valeur binaire de la température ambiante.
Ce point a été décisif dans le choix du composant, car c'est en général le microcontrôleur qui se charge de cette conversion qui demande beaucoup plus d'énergie (un peu plus de 1mA contre seulement 75µA pour le capteur de température). De plus ce composant offre un mode de fonctionnement spécialement dédié aux applications à basse consommation d'énergie.
(le digramme block et le pinout* du capteur de température sont en annexe).

#### 1.3.2.1. Les registres

Le capteur de température n'a pas d'espace mémoire organisé sous forme de tableau comme la mémoire M24LR64. Il fonctionne avec le principe de registre ce sont des petites zones mémoires de quelques bits, prévues pour recevoir un type de données précises.

L'utilisateur peut accéder à ces registres grâce à un pointeur de registre. Ce pointeur peut prendre quatre valeurs chacune d'entre elles donne accès à un des registres.
Dans l'application datalogger, je n'utilise que deux de ces registres :

- **le registre de configuration** (8bits): Il me permet de configurer les options d'acquisition de température, comme la résolution (9-12bits) ou activer le mode One-Shot qui permet de mettre le composant en stand-by dès qu'une acquisition est terminée, utile pour les applications à basse consommation d'énergie.
- **le registre de température** (16 bits): il contient la dernière température acquise, il n'est accessible qu'en lecture, il est impossible d'y écrire quelque chose.

#### 1.3.2.2. Le format de la température

La température est représentée par une valeur codée sur 9,10,11 ou 12 bits selon la résolution choisie, ce qui revient à choisir une précision de 0,5°C à 0.0625°C avec un temps de conversion allant respectivement de 85ms à 680ms.
Pour l'application datalogger, il s'agit de consommer le moins possible d'énergie, et le codage sur 9bits est le plus rapide, j'ai choisi de rester sur une précision de 0.5°C. La conversion se fait suivant le tableau (disponible en annexe) :

#### 1.3.2.3. Les commandes I2C

Le STTS75 est connecté comme esclave sur le bus I2C , c'est-à-dire qu'il ne fait que recevoir des ordres, il n'en donne aucun. Pour être identifié par les autres composants sur le bus I2C, le STTS75 a un identifiant qui est : «1001 0000» en valeur binaire ou 0x90 en valeur hexadécimale.

Les commandes I2C sont envoyées par le Maître, pour l'esclave. Dans l'application datalogger le STTS75 va donc recevoir des commandes I2C venant du Microcontrôleur STM8L.

Dans cette application j'ai besoin du capteur de température pour deux types de commandes I2C :

- Acquisition de la température
- Lecture de la température acquise

Voici, ci-dessous un exemple des commandes I2C que le STTS75 va recevoir de la part du microcontrôleur (STM8L).

**Exemple d'une acquisition de la température** :




*26.figure : commande I2C acquisition d'une température*

Comme pour la communication avec la mémoire, j'ai utilisé la bibliothèque I2C standard du microcontrôleur «stm8l10x_i2c.c» pour construire un fonction générique «aquisition_température» je l'ai appelée :

```
void I2C_SS_Config(uint16_t ConfigBytes)
```

ConfigBytes : est l'information de configuration qui permet de faire une acquisition.

Dans mon application, le microcontrôleur accède au registre de configuration ou il configure une acquisition en mode One-Shot* avec une résolution de 9 bits.

*One-Shot* : opération ponctuelle.*

**Exemple d'une lecture de la température acquise**:




*27.figure : commande I2C lecture de la température acquise*

On retrouve ci-dessous le prototype de la fonction que j'ai créé pour lire la température acquise, Le code de cette fonction est disponible en annexe dans la partie «i2c_ee.c».

```
void I2C_SS_BufferRead(uint8_t* pBuffer, uint8_t Pointer_Byte, uint8_t NumByteToRead)
```

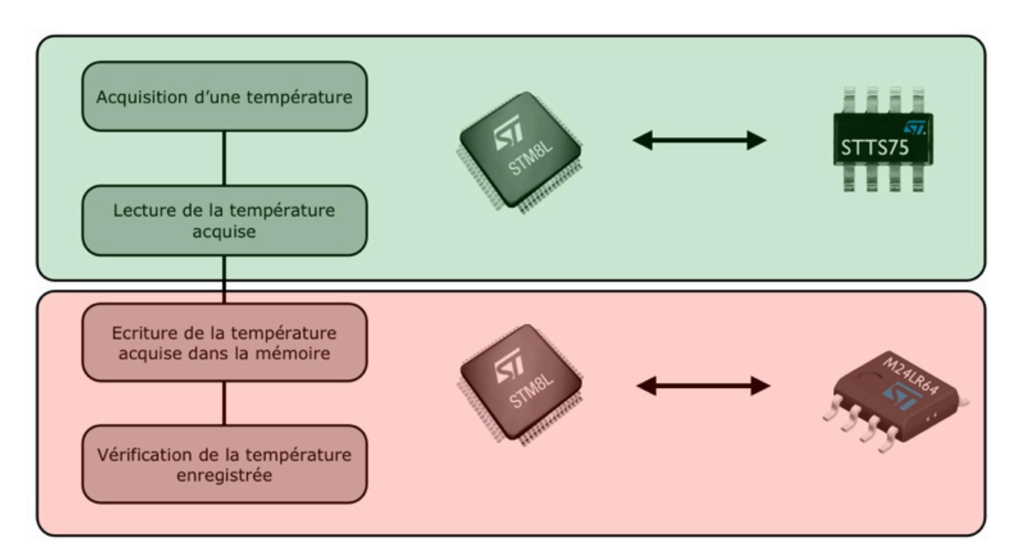
pBuffer : est la température lue.
Pointer_Byte : est l'adresse à laquelle lire l'information
NumByteToRead : est la longueur de l'information à lire (nombre d'octet)

Le microcontrôleur accède au registre de température où il commence une lecture des 16 bits du registre pour récupérer la valeur de la température acquise.

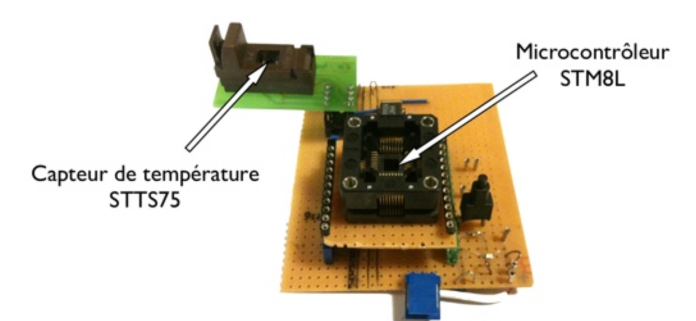
#### 1.3.2.4. Difficultés rencontrées

Bien que le capteur de température soit un composant STMicroelectronics j'ai rencontré quelques difficultés pour me procurer ces composants, car ils ne sont pas réalisé sur le site de Rousset, mais à STMicroelectronics Carrollton au Texas. Les commandes étant toujours bloquées, j'ai passé une commande d'échantillons gratuit, pour contourner le problème. Cette demande à été évidemment acceptée et j'ai pu commencer à travailler avec le capteur de température une semaine plus tard.

#### 1.3.2.5. Validation




La validation de la communication entre le microcontrôleur et le capteur s'est faite sur la même carte prototype que pour valider la communication avec la mémoire, où j'ai ajouter le capteur de température STTS75. Pour valider un échange complet avec les trois composants, cette même carte prototype a été utilisé le but a été de faire fonctionner l'algorithme ci-contre.




*28.figure : prototype pour la validation communication I2C*

### 1.3.3. Microcontrôleur [STM8L]

**Rôle** : Il contrôle l'application, c'est lui qui permet d'établir la communication entre les composants et de les alimenter. Il est à la base de l'autonomie du système.



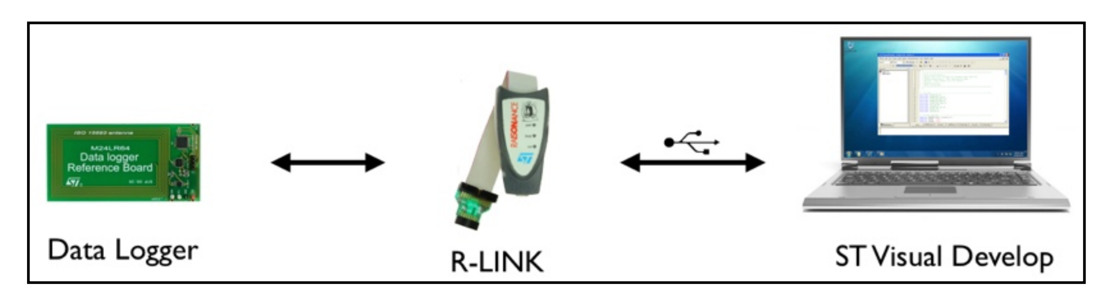

**STM8L** Microcontrôleur 8 bits Ultra basse consommation d'énergie.

- Alimentation : de 1.65 à 3.6V
- faible consommation de courant
  - ≈1µA en mode Active-Halt
  - ≈500µA en mode Run
- compatible avec l' interface I2C
- gamme de fonctionnement : -40°C à 125°C
- mode de fonctionnement basse consommation d'énergie

Le STM8L est
un microcontrôleur spécialement conçu pour des applications à basse consommation d'énergie. (le diagramme block et le pinout du microcontrôleur sont en annexe).

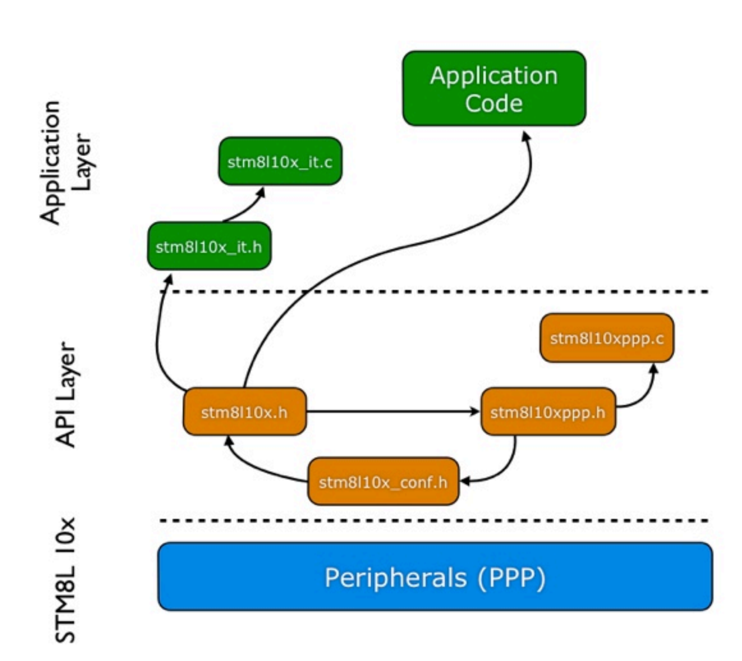
#### 1.3.3.1. Le matériel

Comme expliqué précédemment (voir partie 5 Datalogger) le microcontrôleur exécute un programme. Pour programmer un microcontrôleur il faut un matériel adapté.
Pour le STM8L j'ai utilisé :
- un logiciel PC développé par STMicroelectronics «ST_Visual_Develop®» pour écrire le code.
- le matériel R-LINK pour transférer le code dans le STM8L.




#### 1.3.3.2. Description des librairie

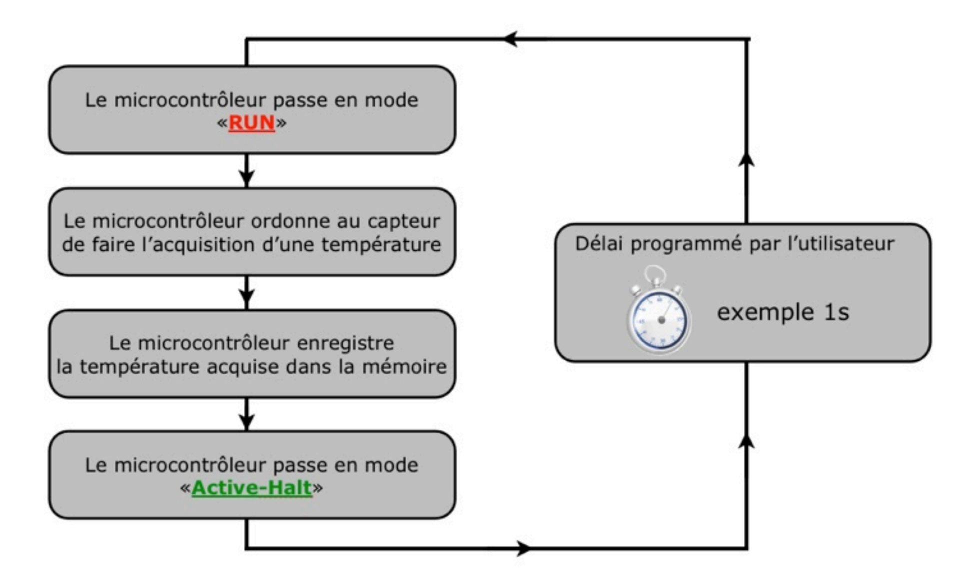
Je développe la routine du datalogger en langage C dans la partie nommée «Application code» dans le schéma ci-contre. Ce schéma montre la relation entre les fichiers des librairies standards.




*29.figure : description des couches logicielles du STM8L*

### 1.3.3.3. La routine

Dés qu'il est alimenté, le microcontrôleur exécute le programme qu'il contient. Ce programme est expliqué ci-dessous sous forme de routine : une suite d'actions répétées en boucle.




Dans l'application datalogger, le programme que j'ai implémenté est très simple à comprendre. Voir ci-contre l'algorithme simplifié du programme microcontrôleur.

Ainsi l'application fonctionne en permanence et enregistre la température ambiante à intervalles réguliers.

Pour ne pas consommer d'énergie inutilement, je fais passer le microcontrôleur en mode de basse consommation d'énergie «**Active-Halt**» pendant l'intervalle déterminé par l'utilisateur (ici une seconde).

Pour mieux comprendre le principe de ce «mode de fonctionnement» du microcontrôleur, il faut regarder ce qui se passe en terme de consommation d'énergie.

Dans le chronogramme ci-contre, on voit en rouge le mode «**RUN**» le microcontrôleur est configuré dans ce mode quand il exécute des actions (lecture ou écriture de la mémoire, ou du capteur de température).
Précisément, le microcontrôleur est en mode «**RUN**» quand il récupère la température ambiante et qu'il l'enregistre dans la mémoire. Ces opérations (cumulées), prennent environ 150 ms et nécessitent un courant de 500µA.
Une fois ces opérations terminées le microcontrôleur passe en mode d'économie d'énergie «**Active-Halt**» pendant une seconde. On voit ce mode repéré en vert sur le schéma, la consommation se résume à quelques microAmpères.

En résumé, dans cette routine : le système ne consomme de l'énergie que lorsqu'il en a besoin.

### 1.3.3.4. le programme

La routine est une version très simplifiée du programme en lui même. Si je veux que cette routine s'exécute correctement, il faut que mon programme soit le plus robuste possible, qu'il puisse faire fasse à toute les situations. Je vais donc vérifier un maximum de paramètres comme :

- combien de températures sont enregistrées dans la mémoire ?
- quel est la fréquence d'acquisition choisie par l'utilisateur ?
- le système est-il en état de marche ou à l'arrêt ?
- que fait le système lorsque la mémoire est pleine ?

Toutes ces informations se trouvent stockées dans les deux premiers block de la mémoire M24LR64 (voir figure ci dessous) les éléments sont détaillés dans la partie 1.3.1 «Mémoire double interface» partie STM8L Source code «main.c».

| numéro secteur | RF block adresse | i2c octet adresse | bit [31:24] | bit [23:16] | bit [15:8] | bit [7:0] |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | RFU | Delay | Overwrite | Status |
| 0 | 1 | 4 | RFU | RFU | Nb Temp[1] | Nb Temp[0] |

Le microcontrôleur peut donc aller consulter ces cases mémoires et réagir en fonction de leur valeurs, il peut aussi modifier leur contenu pour mettre à jours les données.

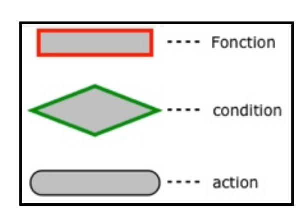
L'algorithme ci-dessous est la représentation du code avec lequel j'ai programmé le microcontrôleur, il aide à comprendre l'échange permanent avec la mémoire double interface. Il détaille toutes les opérations réalisées par le microcontrôleur (une partie du code est disponible en annexe ).

**Fonction principale :**

Alimente bus I2C
Alimente mémoire
Première exécution ?
oui
stop_acquisition
non
Lire l'octet Status dans la mémoire
Valeur de l'octet Status
ELSE
UPDATE
START
RUNNING
acquisition_update
start_acquisition
acquisition_running
Désalimenter capteur de température
Désalimenter mémoire
Désalimenter bus I2C
Passer en mode basse consommation
Pause de x seconde(s)

*30.figure : algorithme principal microcontrôleur*

L'algorithme ci-dessus se lit de la manière suivante :




**Les rectangles rouges** lancent l'exécution d'une autre fonction (d'un autre algorithme) une fois cette autre fonction terminée l'algorithme principal reprend là ou il s'était arrêté.

**Les losanges verts** posent une question. La réponse oriente le déroulement de l'algorithme.

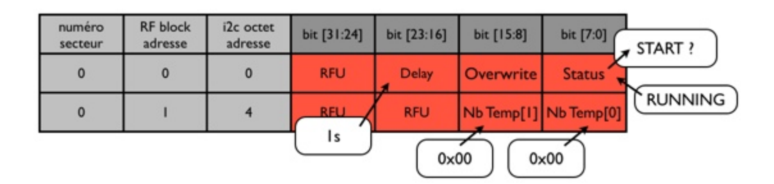
**les ovales noirs** sont des actions directes du microcontrôleur.

**Fonction start_acquisition :**

start_acquisition
Ecrire 1s dans l'octet Delay de la mémoire
Lire l'octet Delay de la mémoire
Configurer la fréquence d'acquisition (AWU*)
Ecrire RUNNING dans l'octet Status de la mémoire
Ecrire 0x0000 dans l'octet NbTemp de la mémoire

on voit la correspondance, entre l'exécution du programme du microcontrôleur, et les actions dans l'espace mémoire.
Cette fonction start_acquisition est exécutée dés que la valeur de l'octet Satus = START.
Si c'est le cas, cela signifie que l'utilisateur souhaite commencer une acquisition, et donc il faut paramétrer les valeurs par défaut :

- Nombre de température acquise = 0x0000
- L'acquisition a commencé donc l'octet Status = RUNNING
- La fréquence d'acquisition par défaut est de 1 seconde

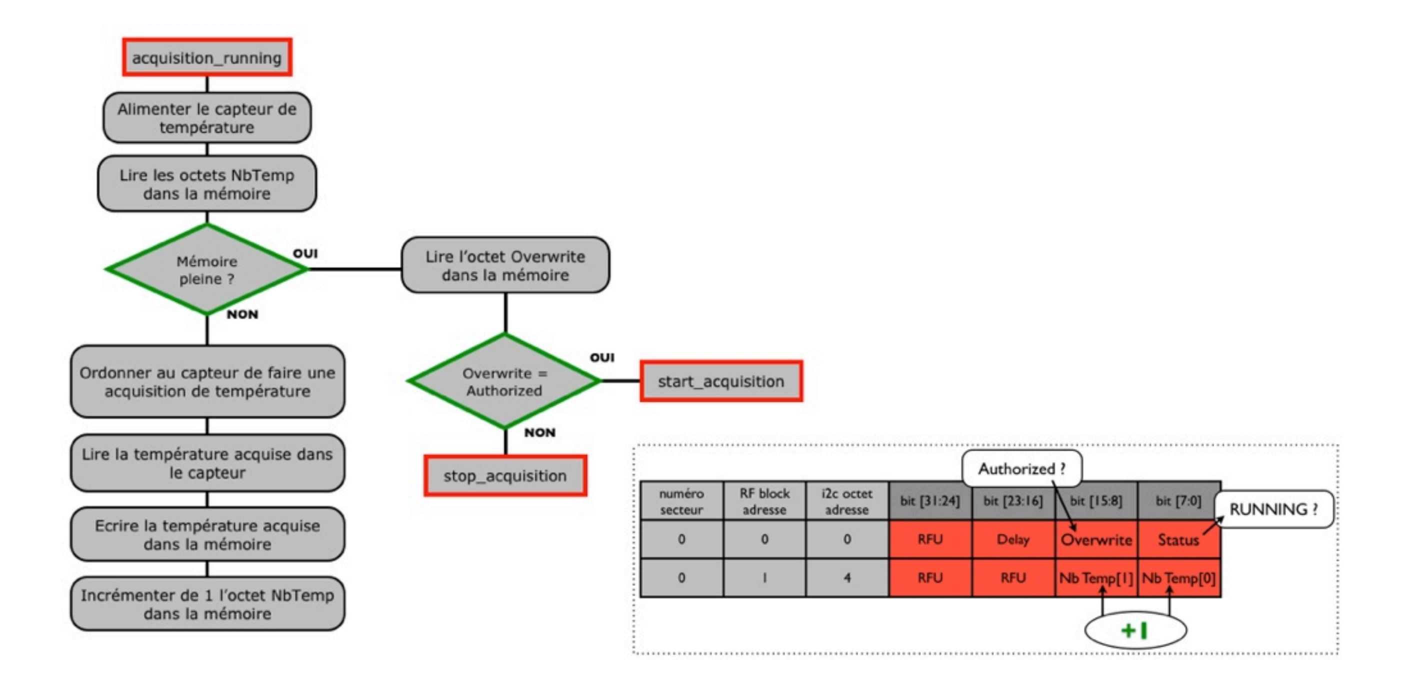



Le code de la fonction est disponible en annexe partie STM8L source code «main.c».

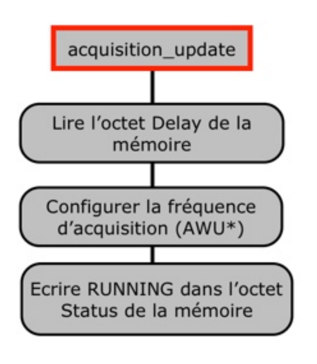
**Fonction acquisition running :**

Cette fonction acquisition_running est exécutée dés que la valeur de l'octet Satus = RUNNING.
C'est le cas dès qu'une acquisition est lancée, c'est cette fonction qui va être exécutée en boucle jusqu'à un changement de l'octet Status.
Il faut vérifier à chaque enregistrement de température :

- Si la mémoire est pleine
- Si l'utilisateur a autorisé un écrasement des données en cas de mémoire pleine.
- Ajouter +1 dans le nombre de température acquise.



**Fonction acquisition_update** :




Dès que l'utilisateur choisit de changer la fréquence d'acquisition, cette fonction s'exécute, quel que soit la valeur de l'octet Satus.
Il faut :

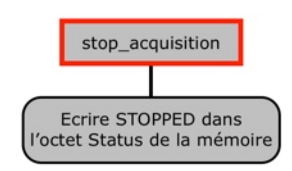
- Lire la valeur de Delay que vient de changer l'utilisateur
- Configurer la nouvelle fréquence dans le microcontrôleur.
- L'octet Status prend la valeur RUNNING ce qui permet de changer la fréquence sans avoir à arrêter l'acquisition en cours.

nouvelle valeur

| numéro secteur | RF block adresse | i2c octet adresse | bit [31:24] | bit [23:16] | bit [15:8] | bit [7:0] |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | RFU | Delay | Overwrite | Status |
| 0 | 1 | 4 | RFU | RFU | Nb Temp[1] | Nb Temp[0] |

RUNNING

**Fonction stop_acquisition :**




Dès que l'utilisateur choisi d'arrêter une acquisition en cours, ou lorsque que la mémoire est pleine sans autorisation «Overwrite » cette fonction s'exécute, quel que soit la valeur de l'octet Satus.
Il faut :

- Ecrire la valeur STOPPED dans l'octet Status.

Authorized ?

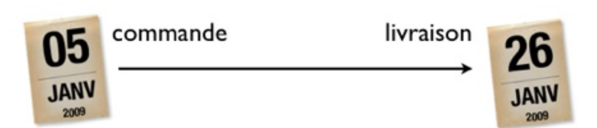
| numéro secteur | RF block adresse | i2c octet adresse | bit [31:24] | bit [23:16] | bit [15:8] | bit [7:0] |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | RFU | Delay | Overwrite | Status |
| 0 | 1 | 4 | RFU | RFU | Nb Temp[1] | Nb Temp[0] |

STOPPED

### 1.3.3.5. Difficultés rencontrées

Sur un site comme celui de rousset comptant prés de 3000 personnes, lorsque l'on recherche une information précise, on se rend vite compte qu'on ne sait pas exactement à qui la poser. le problème c'est que je n'ai aucun contact de l'équipe microcontrôleur. Dans ce genre de situation je m'en reporte à mon tuteur qui connait surement une personne qui pourrait me renseigner. Je note ensuite ce contact et son poste pour un éventuel futur besoin.

J'ai pris rendez vous avec un ingénieur de l'équipe des applications de la division microcontrôleur Jacky.B, pour récupérer le matériel nécessaire au développement sur STM8L.




En allant au rendez-vous pour récupérer le matériel, je n'avais pas prévu de devoir passer une commande et je pensait le récupérer immédiatement, ça engendre donc un premier décalage du planning.

En janvier 2009 le STM8L n'est pas encore commercialisé, je récupère les échantillons d'une première version qui connait évidemment certain dysfonctionnements.

#### 1.3.3.6. Validation

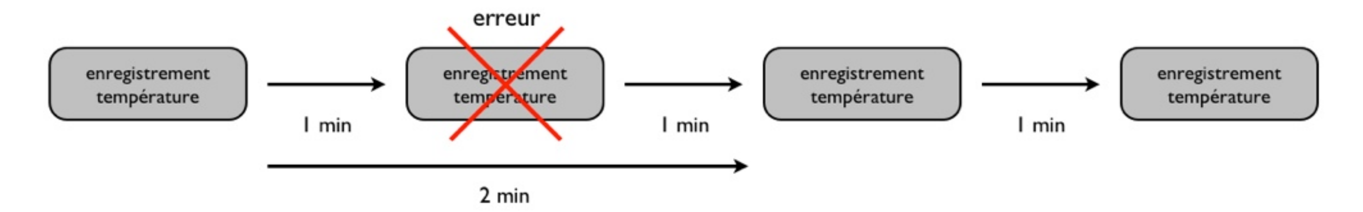
Lors des test de validation je fais tourner la routine basique en boucle en générant des variations de température sur le capteur de température à l’aide d’une bombe à froid,puis grâce à un lecteur RFID je vérifie la cohérence des valeurs enregistrées dans la mémoire double interface.
En faisant ce test j'accède à la mémoire pour la première fois par les deux protocoles (I2C & RF), et je constate un un conflit.




L’arbitrage de la mémoire double interface est très simple il est basé sur le principe «premier arrivé premier servi».
Si l’ I2C et la RF souhaite accéder à la mémoire au même moment, celle-ci traitera la première arrivée et ne traitera pas l’autre.

En effet lorsque je souhaite accéder à la mémoire en I2C alors qu’une communication RF est en cours je constate que la commande I2C n’est pas traitée, mais ce qui est plus embêtant c’est que je constate aussi que le bus I2C lui même se bloque est devient inutilisable après ce genre de conflit. Apres plusieurs tests et vérification, je découvre que le problème provient en fait de la bibliothèque I2C livrée avec le microcontroleur STM8L. En effet ce produit comme je l’ai dit un peu plus haut n’est pas mature et ses bibliothèques non plus.
J’ai donc implémenté une solution permettant de libérer le Bus I2C en cas de conflit (I2C & RF). Elle consiste à générer la condition de «STOP» sur le bus I2C en cas d’erreur. Cette solution résout le problème mais, c’est embêtant car dans ce cas de figure le système n’a pas à enregistré de température quand la communication RF a généré le conflit.

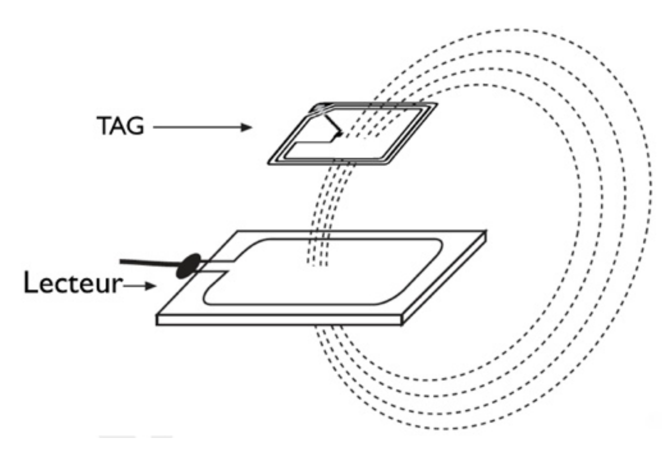
Imaginons que l'utilisateur ai fixé une fréquence d’acquisition d’une minute, si une communication RF perturbe une de ces communications I2C alors, cette erreur génère un délai de deux minutes (voir schéma ci-dessous).




La solution à ce type de problème est d’implémenter un algorithme qui consiste à recommencer une action si elle échoue, mais seulement un certain nombre de fois (pas indéfiniment).
Il ne faut pas oublier que le datalogger consomme principalement de l'énergie lorsqu’il effectue des communications I2C.

## 1.4. L'antenne

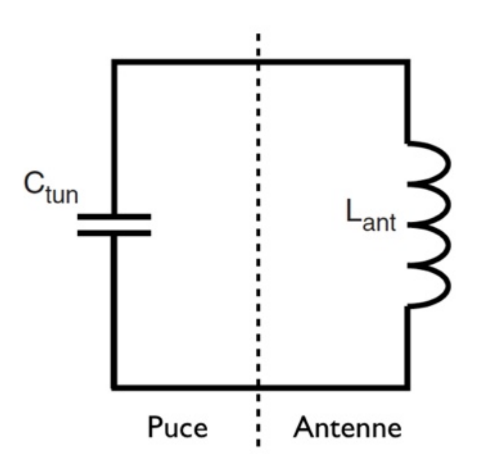
Les TAG RFID extraient leur énergie du champs électromagnétique émit par un lecteur. L'antenne du TAG et du lecteur forment une paire d'inductance couplées (voir schéma ci-dessous).




Si on considère un lecteur du marché fonctionnant correctement, ce transfert d'énergie du lecteur vers le TAG dépend de la précision de la fréquence d'accord de l'antenne du TAG à 13,56MHz.

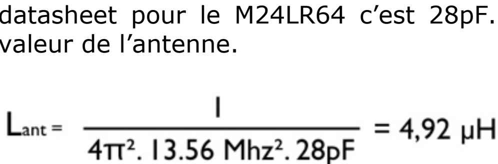
### 1.4.1.Calcul

Je dois donc designer une antenne qui une fois connectée à la puce (mémoire double interface) devra être accordée à 13.56MHz. Pour se faire, il faut connaitre le schéma équivalent simplifié ci-dessous.




On voit ci-contre que la puce est considérée comme sa capacité interne et l'antenne comme son inductance équivalente.

La fréquence de raisonnance d'un circuit LC peut être calculée avec l'équation :



$$F_0 = \frac{1}{2\pi\sqrt{L_{ant} . C_{tun}}}$$

La capacité du produit mémoire est communiquée dans la datasheet pour le M24LR64 c'est 28pF. je peux en déduire la valeur de l'antenne.

$$L_{ant} = \frac{1}{4\pi^2 . 13.56\ Mhz^2 . 28pF} = 4{,}92\ \mu H$$

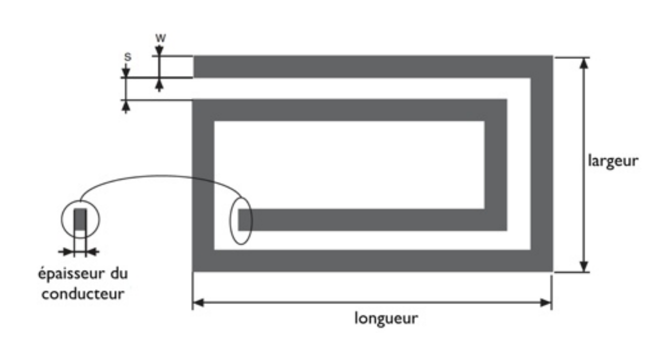
La capacité communiquée interne de la mémoire peut avoir un pourcentage d'erreur, alors je décide de réaliser trois prototypes d'antenne à ± 5% de la valeur calculée soit :
4,67µH / 4,92µH / 5,16µH.

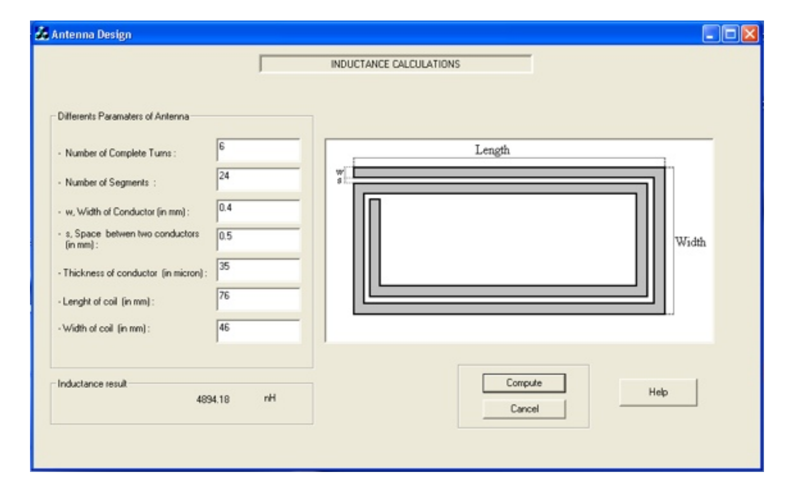
### 1.4.2. Design

Il me reste à designer ces trois antennes. J'utilise un logiciel développé par STMicroelectronics «antenne.exe» qui compile des données d'entrées (voir ci-dessous) pour calculer une valeur d'inductance de l'antenne.

Je dois renseigner :

- le nombre de tour
- le nombre de segment
- **w** la largeur de la piste (mm)
- **s** l'espace entre les piste (mm)
- l'épaisseur du conducteur (µm)
- la longueur de l'antenne (mm)
- la largeur de l'antenne (mm)






Le logiciel me donne une valeur théorique de l'inductance en fonction des paramètre d' entrée.

Pour la longueur et la largeur je me base sur un format ISO celui des carte de transport RFID soit : (45mm x 75mm), ensuite je m'arrange pour m'approcher le plus possible des valeurs calculées plus haut.

| Antenne | inductance théorique | inductance logiciel | nombre de tours | largeur conducteur | espace entre conducteur |
|---|---|---|---|---|---|
| 1 | 4,67µH | 4,62µH | 6 | 0,5 mm | 0,5 mm |
| 2 | 4,92µH | 4,89µH | 6 | 0,4 mm | 0,5 mm |
| 3 | 5,16µH | 5,12µH | 6 | 0,4 mm | 0,4 mm |

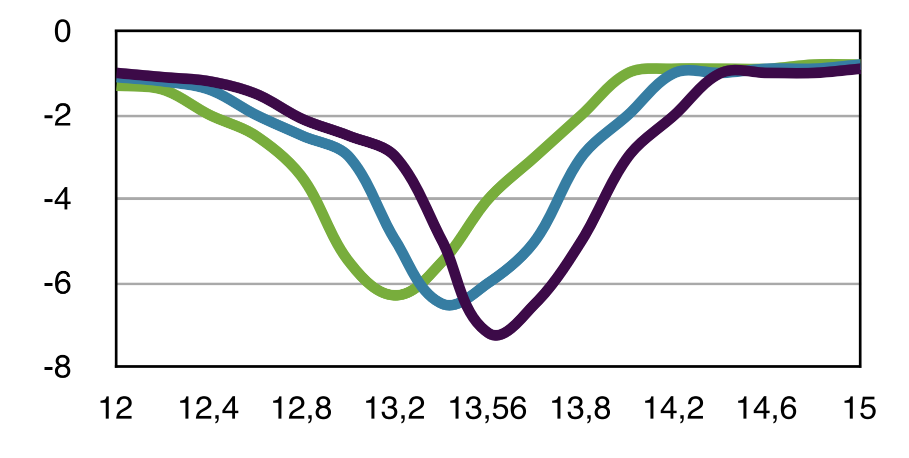
### 1.4.3. Validation

Je fais réaliser les trois prototypes par le sous traitant Synergie-CAD. A la réception des antenne je les connecte aux composants M24LR64 et mesure leurs fréquences d'accord.
La mesure de fréquence d'accord se fait grâce à un analyseur de réseau (Agilent HP 8712ET) et une antenne boucle (loop probe 7405-901).




L'analyseur de réseau émet un signal de puissance -10dB (qui est le champs minimum pour alimenter la puce) sur un domaine de fréquence allant de 12MHz à 15MHz. Et analyse le

Les résultats des trois antennes sont exprimés ci-dessous, sous forme de courbe de sur lesquelles on distingue nettement la fréquence d'accord.




Je garde donc le design de l'antenne 1 qui est est accordé à 13,56MHz.

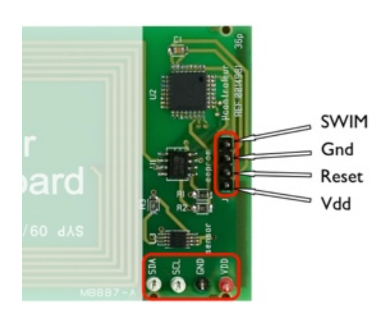




## 1.5. Réalisation de la carte

A cette étape, l'étude du datalogger et de l'antenne sont terminées et validées sur la carte «prototype». Je peux maintenant passer à une réalisation plus industrielle, mais le matériel à disposition à STMicroelectronics ne le permet pas.
L'unité RFID a pour habitude de travailler avec un sous-traitant nommé Synergie-CAD® pour la réalisation de cartes comme celle du datalogger.
Je suis entré en contact avec cette entreprise, pour travailler sur une version industrielle de la carte. J'ai inviter Mr Eric.V de Synergie-CAD à mon bureau où je lui ai fourni les informations suivantes :

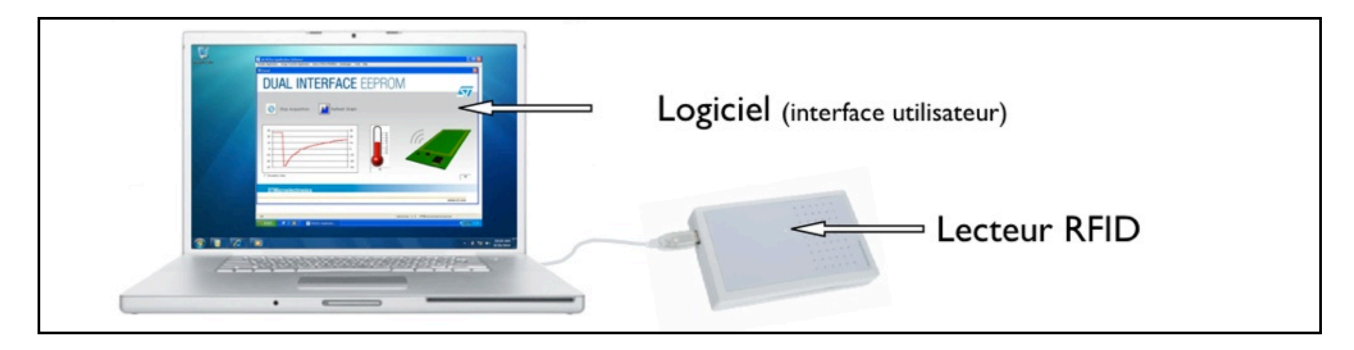
- le schéma de câblage (voir figure x en annexe).
- des recommandations sur l'antenne, les dimensions (45mm x 75mm) le nombre de tour, l'espacement entre les piste, la largeur des pistes. Et la référence de l'antenne qu'ils ont réalisé précédemment et que j'ai retenu comme étant la meilleure.
- l'emplacement des composants et de la batterie.
- le texte pour la sérigraphie («M24LR564 Datalogger Reference Board»)
- le choix des connecteurs, et des points test.




La réunion se déroule comme un échange, avec beaucoup de questions et de remarques. J'ai insisté sur le fait que cette carte devrait être assez «ouverte» avec notamment l'implantation de points test, idéaux pour vérifier par exemple les signaux du bus I2C (voir figure ci-contre).
Le client doit pouvoir s'appuyer sur ce qui est déjà fait pour développer sa propre application, il faut donc lui faciliter l'accès à ce genre de signaux.
J'ai donc décidé également d'implanter un connecteur pour pouvoir programmer et reprogrammer le microcontrôleur à souhait, aussi bien pour débugger mon application avant de la livrer, que pour permettre au client de modifier facilement le programme existant.

# 2. Logiciel interface utilisateur

J'ai entièrement développé le logiciel qui fait office d' IHM*, il s'installe sur un PC équipé d'un système d'exploitation Windows. Il est développé pour contrôler un lecteur RFID connecté en USB à l'ordinateur.
*IHM* : Interface Homme Machine*




*31.figure : description du matériel pour la partie interface utilisateur*

**Le Logiciel** : Il permet de contrôler le lecteur RFID pour démarrer / arrêter le datalogger et télécharger les températures.

- J'ai conçu le logiciel d'interface utilisateur en langage C et Visual basic.

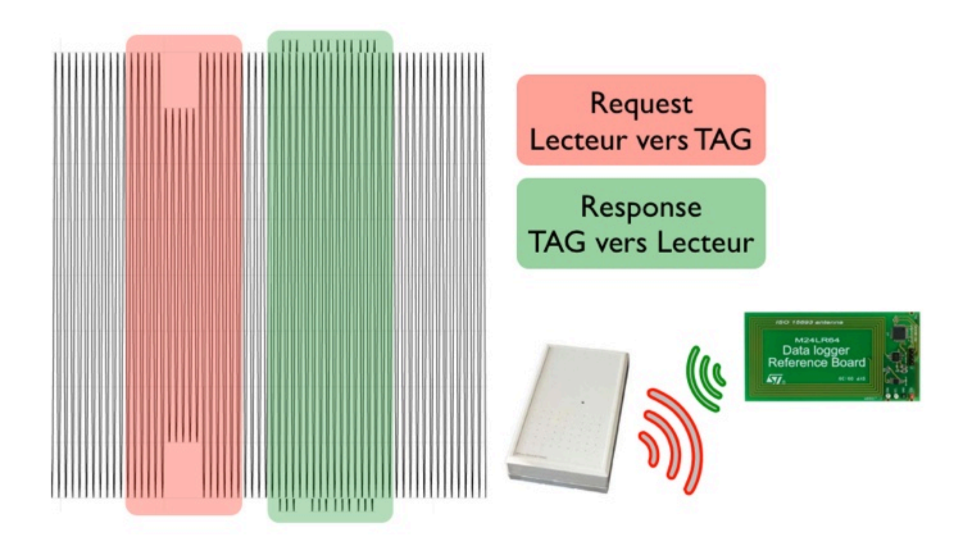
**Le lecteur RFID** : Il permet de communiquer avec la mémoire double interface (M24LR64) présente sur datalogger.

- J'ai utilisé les commandes protocole ISO 15693 pour communiquer la mémoire double interface.

## 2.1. Interface de communication

### 2.1.1. Le protocole RF [ISO15693]

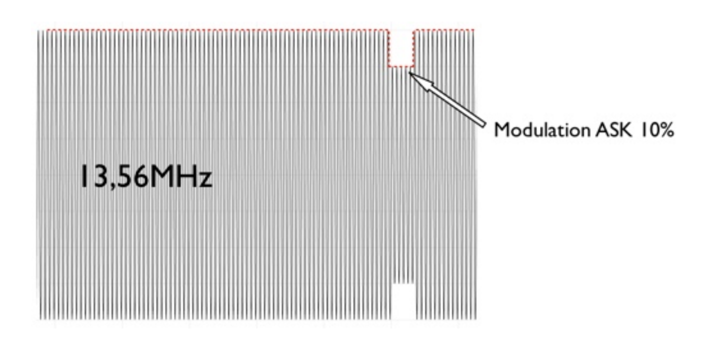
La norme ISO15693 est un standard international décrivant les caractéristiques d'une interface RF entre un lecteur et un TAG. C'est une communication bi-directionnelle, le lecteur envoi une requête, et le TAG renvoi une réponse.




Le transfert d'énergie est effectué par le couplage des antennes du lecteur et du TAG. Le champs électromagnétique émit par le lecteur alimente le TAG. La fréquence de ce champs électromagnétique est de 13,56MHz.

### Requête [du Lecteur vers le TAG]

Le lecteur utilise la modulation d'amplitude ASK 10% ou 100% pour communiquer. Le TAG doit être capable de comprendre les deux. On distingue bien les deux niveaux logique (1 ou 0). Suivant le codage (1 parmi 256) ou (1 parmi 4) les débits sont respectivement de 1,65Kbits/s et 26,48Kbits/s.

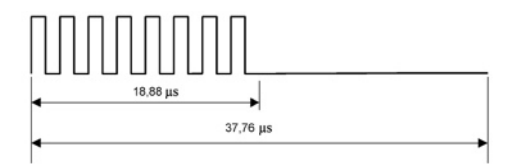



L'information envoyée du lecteur vers le TAG est appelé REQUEST. Elle suit le format de trame suivant :



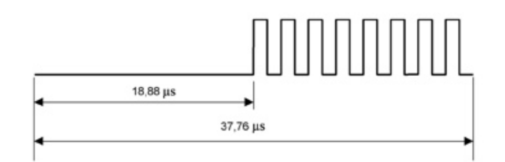

- **[START & END OF FRAME]** La trame RF est délimitée par une condition de départ et une condition d'arrêt, qui délimitent le début et la fin de la trame.
- **[Flags]** Les flags indiquent :
  - le protocole radio physique utilisé (profondeur de modulation, le débit, le codage le nombre de sous porteuse etc...).
  - si je souhaite adresser tous le TAG dans le champs ou seulement un spécifique
- **[Command Code]** : contient le code de la commande à envoyer au TAG.
- **[Parameters]** : contient des informations relatives à la commande (dans le cas d'une écriture il contient par exemple l'adresse à laquelle écrire).
- **[Data]** : contient l'information à transmettre au TAG.
- **[CRC]** : ce champs vérifie l'intégrité de la trame.

### Réponse [du TAG vers le Lecteur]



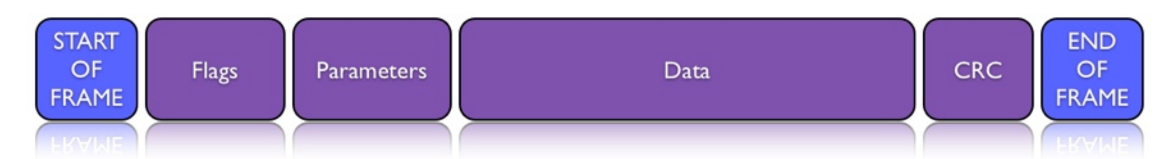

Lorsque le TAG reçoit une requête il renvoi une réponse grâce au codage ci contre.

Le niveau logique 0 est repéré par une suite de 8 pulse à 423,75kHz suivi d'un temps non modulé de 18,88µs.




Le niveau logique 1 est repéré par un temps non modulé de 18,88µs suivi de 8 pulse à une fréquence de 423,75kHz.

La réponse s'établie également sous forme de trame, détaillée ci-dessous :




- **[START & END OF FRAME]** La trame RF est délimitée par une condition de départ et une condition d'arrêt, qui délimitent le début et la fin de la trame.
- **[Flags]** indique un statut erreur ou non il prévient le récepteur des informations qu'il va recevoir (longueur de trame composition)
- **[Parameters]** : contient des informations relatives à la réponse.
- **[Data]** : contient l'information à transmettre au lecteur.
- **[CRC]** : ce champs vérifie l'intégrité de la trame.

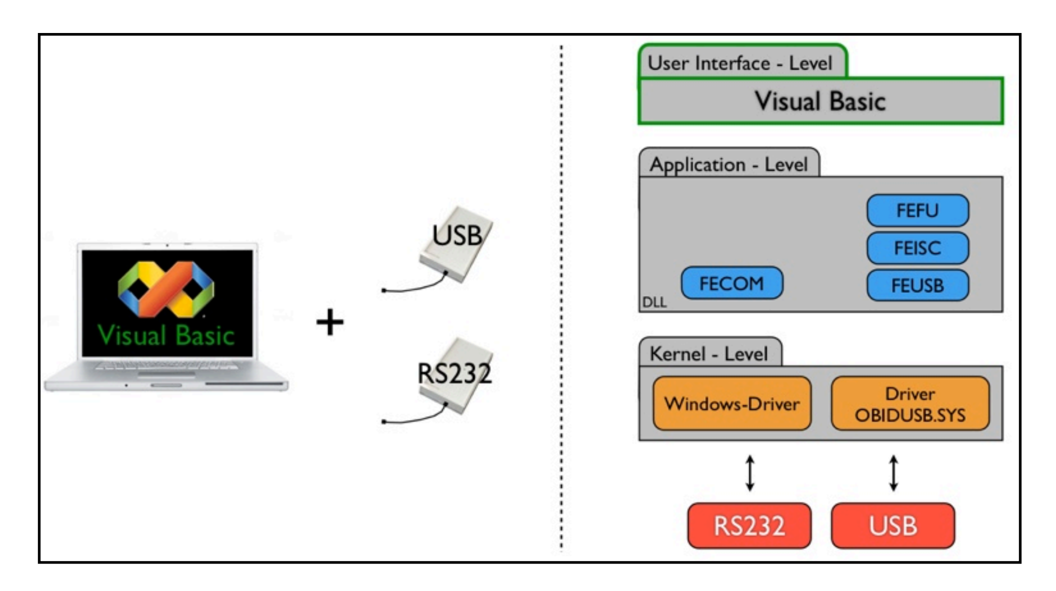
## 2.2. Réalisation

Le but est de créer une interface graphique pour que l'utilisateur puisse utiliser simplement le datalogger.

L'application logiciel est compatible avec 3 lecteurs RFID du marché :
- FEIG USB & FEIG RS232
- ESTAR USB

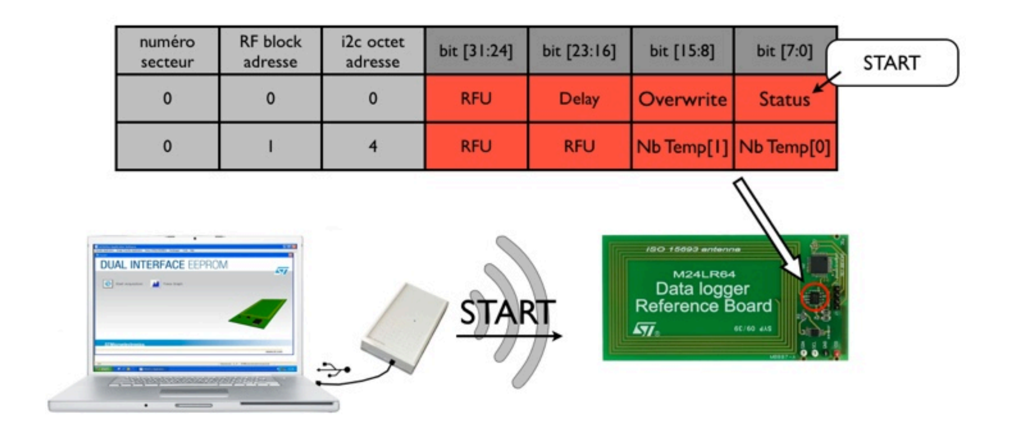
Je vais détailler ci-dessous la mise en place de la solution en prenant l'exemple des lecteurs FEIG qui sont les lecteurs d'un constructeur allemand partenaire de STMicroelectronics. Le constructeur FEIG met à disposition un kit de développement contenant les couches logicielles (Kernel & Application) de bas niveau permettant de développer une application sur Windows (voir illustration ci-dessous).
Le matériel délivré par FEIG (lecteur,DLL,etc.) permet de reconstituer des trames et de les envoyer en RF à la mémoire double interface M24LR64.

Il me reste donc à créer l'interface utilisateur (entouré en vert dans l'illustration ci-dessous). Le logiciel utilisé pour le développement de l'IHM* est Visual basic, c'est un langage particulier qui permet de créer une interface utilisateur graphique.




*32.figure : support logiciel pour Visual basic*

Maintenant que le Firmware est implémenté dans le microcontrôleur, la carte datalogger est autonome et vérifie en permanence la valeur de l'octet Status. Pour démarrer une acquisition il suffit donc simplement :
d'écrire la valeur «START» dans l'octet status de la mémoire double interface, grâce au lecteur RFID (voir illustration ci-dessous).



| numéro secteur | RF block adresse | i2c octet adresse | bit [31:24] | bit [23:16] | bit [15:8] | bit [7:0] |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | RFU | Delay | Overwrite | Status |
| 0 | 1 | 4 | RFU | RFU | Nb Temp[1] | Nb Temp[0] |



*33.figure : écriture de l'octet status via RF*

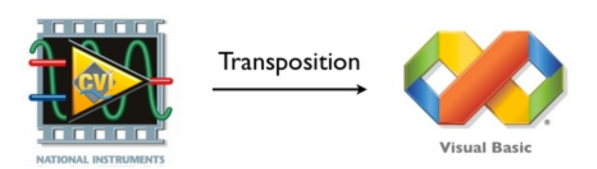
### 2.2.1. Développement Logiciel

J'ai commencé à développer l'interface graphique sur un outil de National Instrument qui s'appelle LabWindow CVI. Au démarrage du projet, j'utilise cet outil car mon tuteur et manager m'y a formé et j'ai déjà une expérience avec cet équipement, mais deux évènements viennent modifier un peu le déroulement du projet :

- en Juin 2009 : Sylvain Fidelis devient manager (n+2) de l'entité RFID.
- en Octobre 2009 : mon tuteur et manager direct (Christophe Mani) quitte le poste qu'il occupe et intègre une autre division. Il est remplacé par Jean-Marie Gaultier qui devient mon nouveau tuteur.

Ces deux changements engendrent donc forcément une modification du type de management. Une fois de plus la gestion de projet en méthode agile permet d'accueillir favorablement le changement, et cette fois il est conséquent.

Pour des soucis de compatibilité et d'harmonisation des langage de programmation au sein de l'équipe je dois passer mon logiciel du langage LabWindows CVI à Visual Basic.
C'est un langage qui m'était inconnu, je me suis donc formé avec l'aide des ingénieurs de l'équipe, et de cours sur internet.




En utilisant le logiciel de développement Visual basic J'ai donc reconstitué les trames pour lire et écrire dans la mémoire, et je les ai associé à des boutons.

De cette façon, il suffit à l'utilisateur de cliquer sur les boutons pour faire effectuer des actions au datalogger.




J'ai donc fait un bouton associé à chaque fonction pour :

- démarrer/arrêter l'acquisition
- afficher les températures sur un graphique.
- Changer la fréquence d'acquisition
- Autoriser l' Overwrite
- connaitre le nombre de températures

Comme le préconisent les méthodes agiles, j'ai livré une version de logiciel fonctionnel à mon manager, pour observer la façon dont il l'utilise et connaitre l'évolution et les modifications qu'il souhaite y apporter.

Le premier retour fut assez clair, puisque le nouveau manager n'avait pas les même attentes que le précédent. Pour lui, il fallait simplifier au maximum l'interface, chaque personne ayant le logiciel devrait pouvoir utiliser les options principales sans même lire le manuel d'utilisation.

J'ai donc réduit le nombre de bouton de la fenêtre principale, et ajouté des animations illustrant les actions générées par l'appui sur les boutons.

Le rendu visuel est important pour vendre ou présenter le produit, et lors d'une démonstration le fait de minimiser le nombre de boutons appui l'idée que l'on souhaite faire passer au client en terme de simplicité d'utilisation.

Je décide donc d'opter pour une interface avec un nombre de bouton réduit et met en place quelques astuces comme : le bouton start et le bouton stop ne sont plus q'un seul bouton qui change d'apparence.
Je fait disparaitre les bouton dès que leurs actions ne sont plus utilisables et réapparaitre lorsqu'elles le redeviennent.

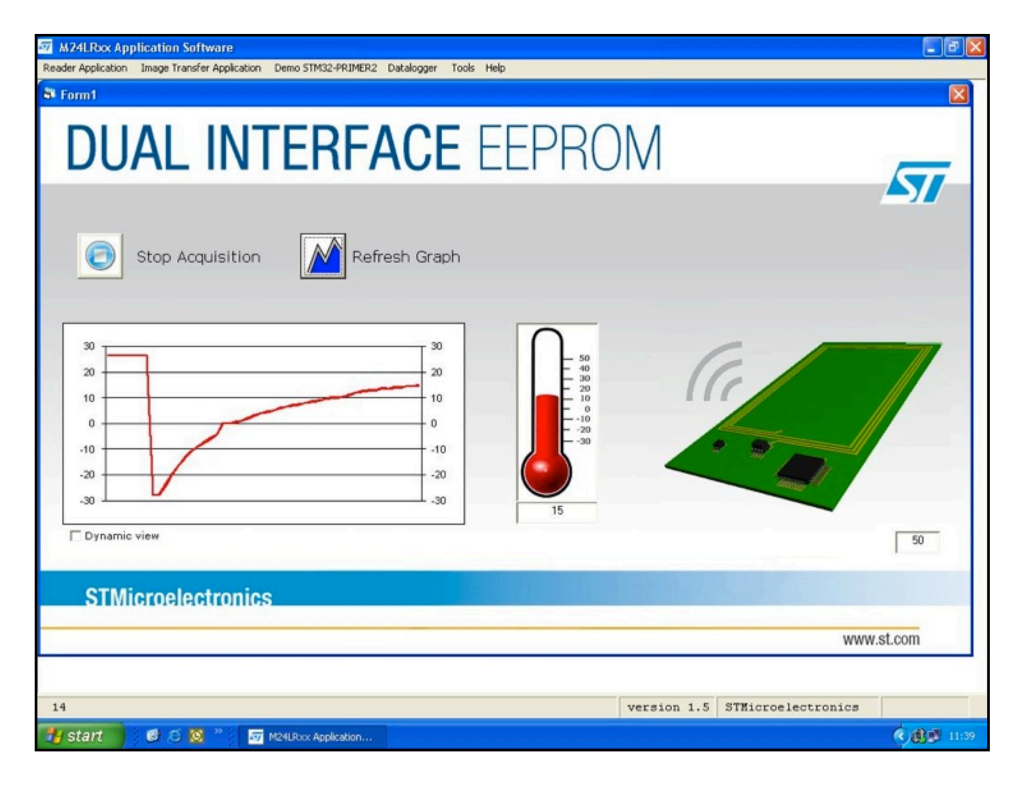
Mon tuteur académique Mr Laurent Freund m'a lui aussi donné quelques conseils :

- le bouton «trace graph» générait l'ouverture d'une fenêtre supplémentaire dans laquelle apparaissait un graphique affichant les températures . Il m'a proposé d'intégrer cette fenêtre à la fenêtre principale pour éviter de surcharger l'écran.
- Il m'a aussi donné l'idée d'afficher l'évolution du graphique en direct.

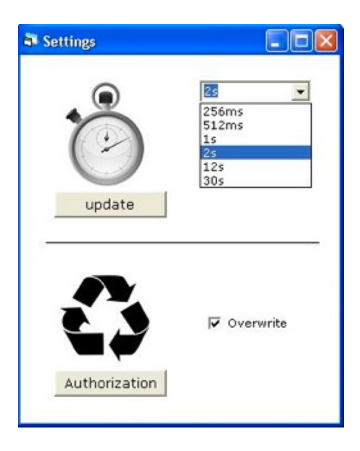
J'ai implémenté ces deux idées dans la version suivante du logiciel et elles ont plu au manager.

### 2.2.2. Version finale

Après plusieurs présentations devant des publics différents et en ayant pris en compte les remarques les plus pertinentes, Voici un aperçu de la version finale de l'interface utilisateur (voir figure ci-dessous).




*34.figure : capture d'écran de l'interface utilisateur datalogger.*




L'utilisateur peut démarrer et arrêter une acquisition de température, ou télécharger les températures acquises afin de les afficher sur un graphique.

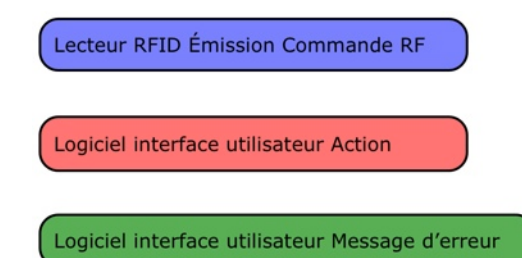
Ce graphique offre une possibilité dynamique :
c'est à dire que si le datalogger reste dans le champ électromagnétique du lecteur RFID, en cochant la case «Dynamic view» en dessous du graphique, j'ajoute en temps réel les températures acquises sur ce même graphique, le thermomètre indique la valeur en degrés et en fonction de celle-ci j'ajoute une illustration météo allant du flocon de neige au soleil.
Au niveau graphique, j'ai ajouté des animations d'ondes lorsqu'une commande RF est envoyée, et je fais clignoter le bus I2C lorsque qu'une acquisition est en cours.



*35.Figure : capture d'écran de la fenêtre d'option interface utilisateur*

### 2.2.3. Algorithmes

Les algorithmes ci-dessous décrivent ce qui se passe lorsque l'utilisateur appui sur les boutons.
J'ai mis en place une légende pour mieux comprendre les différentes actions :




Les actions bleus sont des commandes RFID envoyé par le lecteur par l'intermédiaire de l'interface utilisateur.

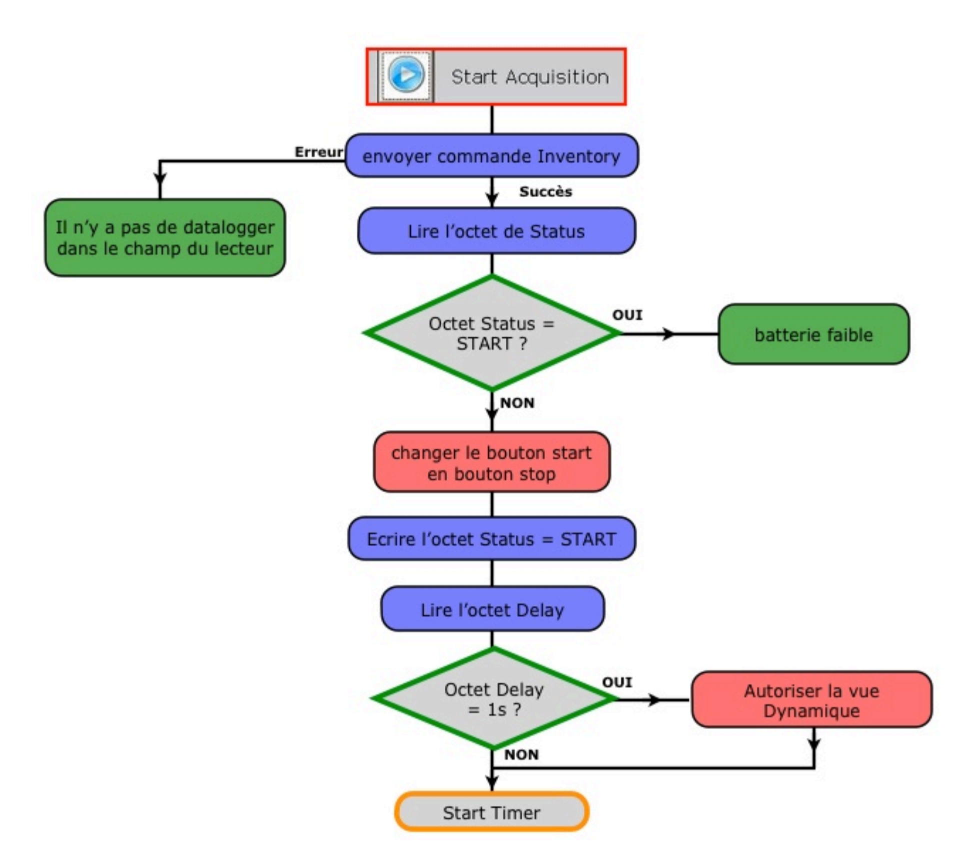
Les actions rouges sont des commandes propres à l'interface utilisateur (modifier l'apparence de la fenêtre).

Les actions vertes sont des messages «pop-up» générés à l'écran.

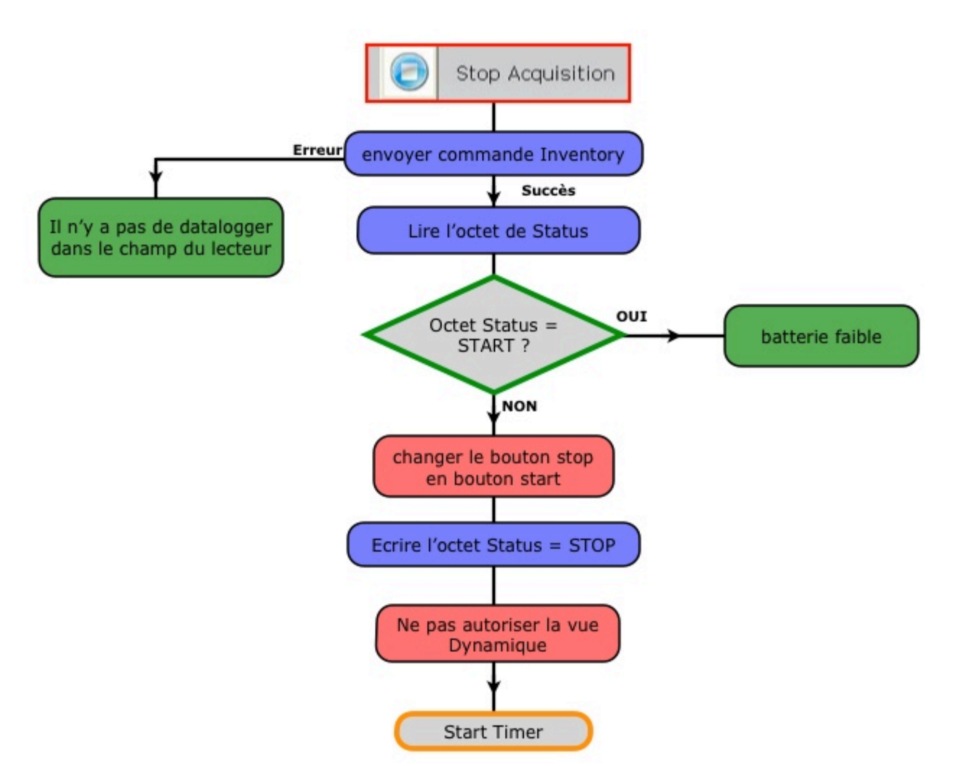
#### 2.2.3.1. Start Acquisition

Lorsque l'utilisateur appui sur le bouton «Start_Acquisition» je dois vérifier la valeur de l'octet Status et y écrire la nouvelle valeur «START» si besoin.
Le bouton se transforme alors en «Stop_Acquisition», une illustration débute et l'algorithme «Timer» (détaillé dans le point 2.2.3.3) se lance.

Dans le cas ou l'octet de Status a déjà la valeur START , cela signifie que la précédente tentative de l'utilisateur n'a pas fonctionné. Le système n'a surement plus de batterie, et dans ce cas je démarre l'illustration correspondante à batterie faible.



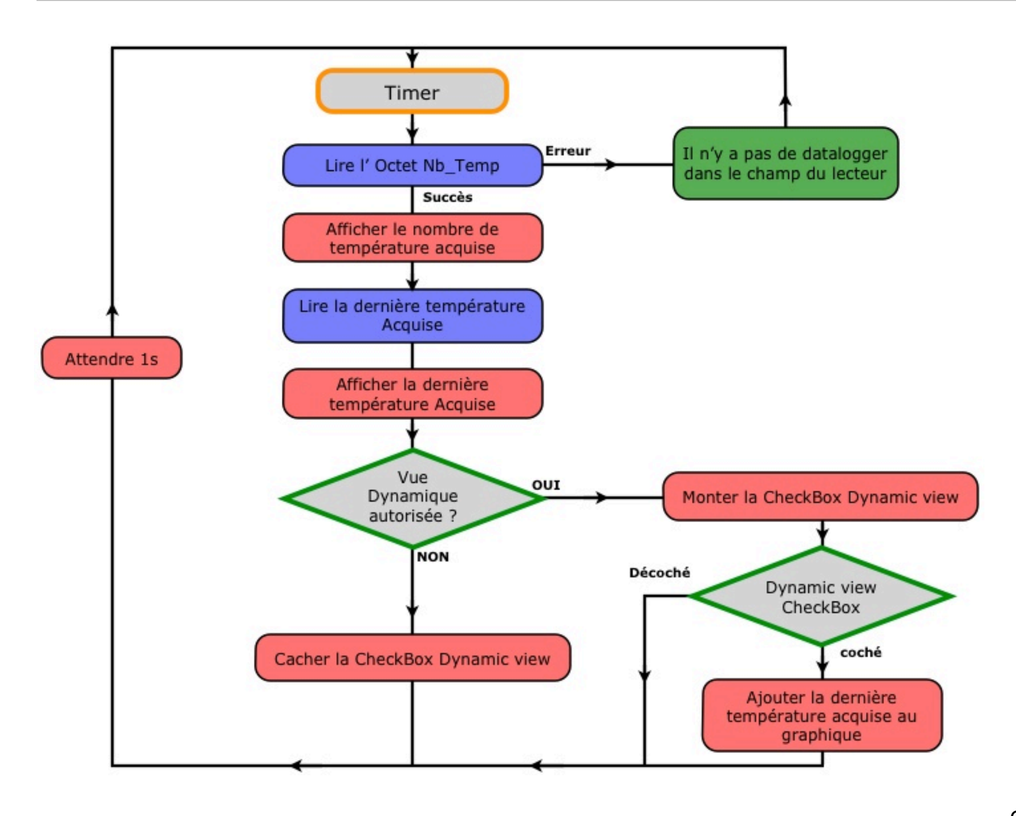

#### 2.2.3.2. Stop Acquisition




Lorsque l'utilisateur appui sur le bouton «Stop_Acquisition» je dois vérifier la valeur de l'octet Status et y écrire la nouvelle valeur «STOP» si besoin.
Le bouton se transforme alors en «Start_Acquisition», une illustration débute et l'algorithme «Timer» (détaillé dans le point 2.2.3.3) se lance.

Dans le cas ou l'octet de Status à déjà la valeur START , ça signifie que la précédente tentative de l'utilisateur n'a pas fonctionné. Le système n'a surement plus de batterie, et dans ce cas je démarre l'illustration

### 2.2.3.3. Timer



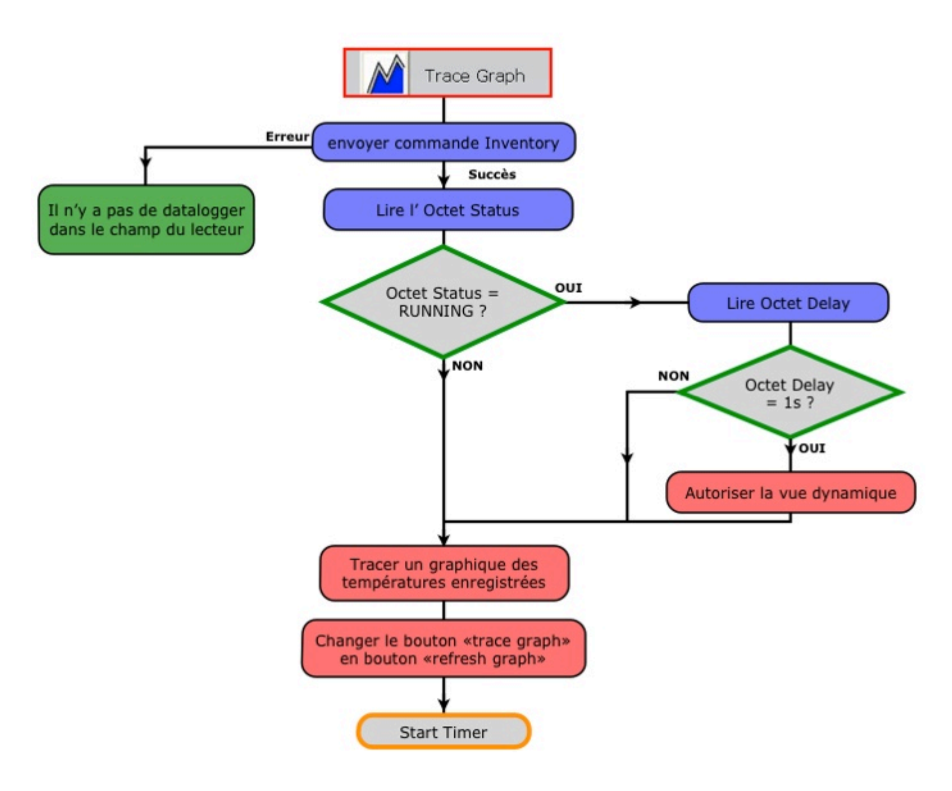

L'algorithme timer est activé et désactivé respectivement par les boutons start & stop acquisition. J'ai choisi de le rafraichir (l'exécuter) toutes les secondes. Il lit les valeurs des octets système dans la mémoire double interface et les affiche sur l'interface utilisateur.

Ainsi on peu connaitre en temps réel le nombre de températures enregistrées. Connaissant ce nombre je peux aller chercher dans la mémoire la dernière température enregistrée et l'afficher à l'écran, ça donne une indication de la température ambiante en temps réel.

J'ai aussi ajouté une option qui permet de voir le graphique des température en mode dynamique (les températures enregistrées s'ajoutent en temps réel au à la suite du graphique).

### 2.2.3.4. trace graph

Le bouton «trace graph» télécharge la totalité des températures enregistrées dans la mémoire double interface et les affiche sur un graphique.



## 3. documentation

Je dois livrer un certain nombre de documents qui seront disponibles dans le reference-design* et sur le site www.st.com :




- **Un guide d'installation**
qui explique la démarche à suivre pour installer correctement le logiciel.

- **Un guide utilisateur**
qui explique comment se servir du logiciel

- **Une Note d'application**
qui donne les détails technique de conception de l'application.

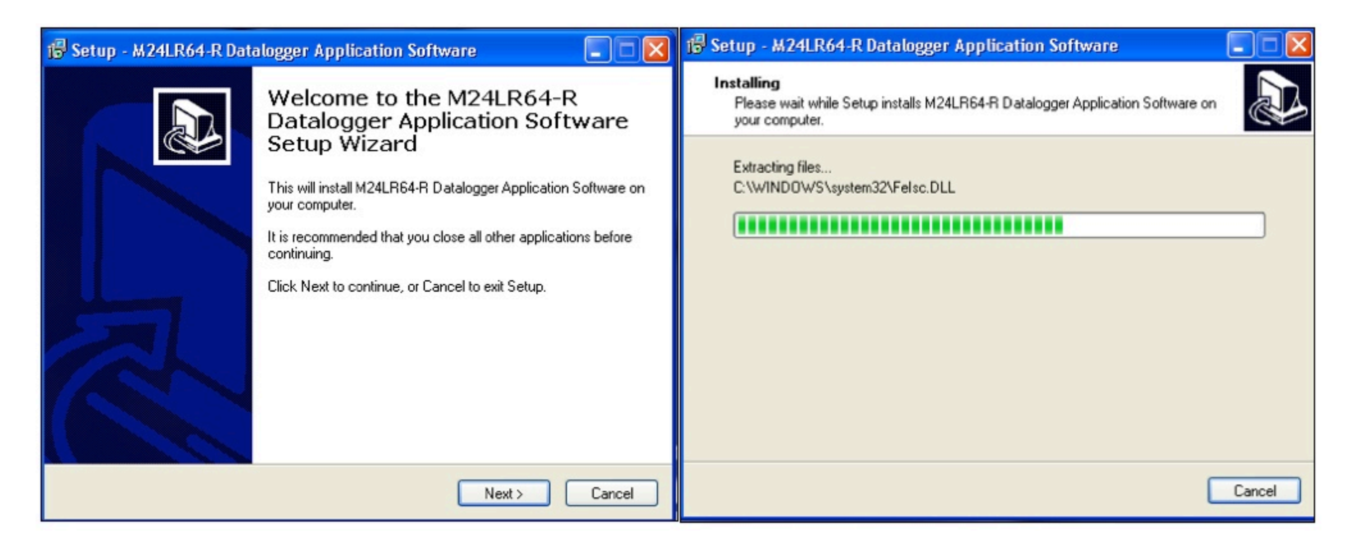
- **Un dossier code source**
qui comprend le code source du logiciel PC et microcontrôleur.

- **Le logiciel**
Le fichier d'installation du logiciel

J'ai rédigé ces documents en anglais uniquement, ce qui m'a permis d'enrichir mon vocabulaire, et m'a entrainé à organiser ma rédaction.

Lorsque je rédige une note d'application ou un guide utilisateur, je travaille avec le service de communication qui vérifie l'ensemble des documents pour les mettre en page avec le standard STMicroelectronics, le document connait plusieurs version que je dois vérifier à mon tour pour valider le contenu.

Pour distribuer le logiciel, j'ai créé un fichier «setup.exe» à l'aide du logiciel «Inno-Setup®», qui installe l'interface utilisateur mais qui copie également les drivers* nécessaires au bon emplacement.

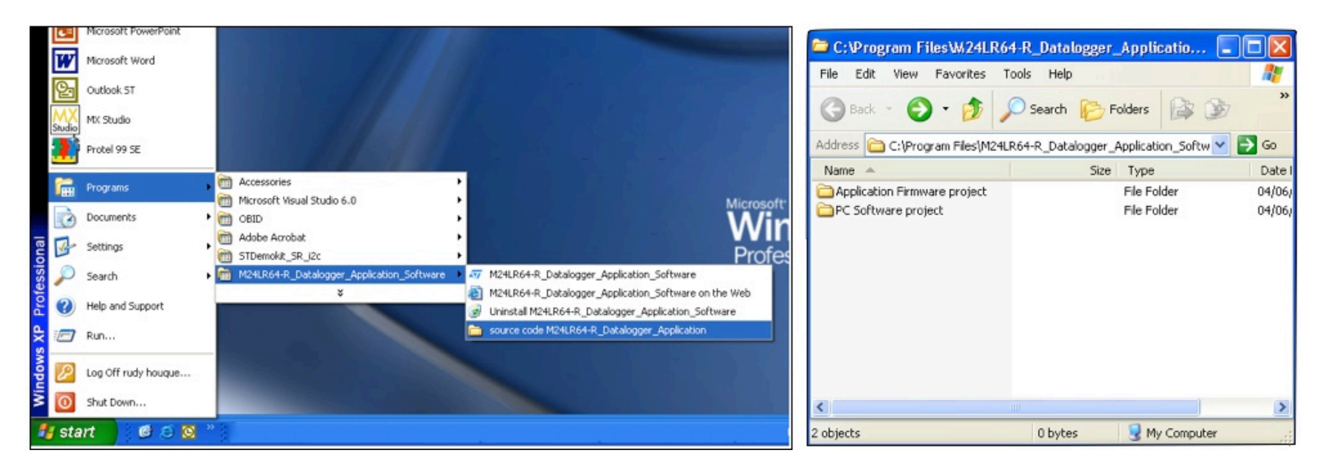



*36.figure : capture d'écran installation du logiciel datalogger.*

driver* : pilote informatique

L'installation créé également un dossier comprenant :
- le code source du microcontrôleur (projet STDV langage C).
- et le code de l'interface utilisateur (projet Visual Basic).

J'ai répertorié les composants installés dans le menu Démarrer > Programmes > M24LR64-R_Datalogger Application software (voir illustration ci-dessous).




*37.figure : Emplacement du package Datalogger Software*

# VI.Bilan

Le projet consistait à concevoir et réaliser un datalogger de température RFID dans le but de promouvoir une nouvelle puce électronique de STMicroelectronics.

## 1. Atteinte de l'objectif

### 1.1. Technique

Le projet a été mené à son terme en respectant les spécifications de départ et en intégrant des améliorations au fur et à mesure le l'avancement. L'application remplit aujourd'hui bien son rôle, celui de démontrer les capacités du produits pour en faire sa promotion.

Si le datalogger été dès le départ voué à n'être commercialisé que sous la forme d'un reference-design*, à la vue du résultat final le manager marketing à décidé de l'utiliser également sous la forme d'un démonstrateur pour lequel j'ai designé une nouvelle version du datalogger dans un boitier en plastique beaucoup plus esthétique. Dans cette nouvelle version j'ai intégré une antenne de plus petite taille (20mm x 40mm), pour persuader plus facilement de la facilité d'intégration.

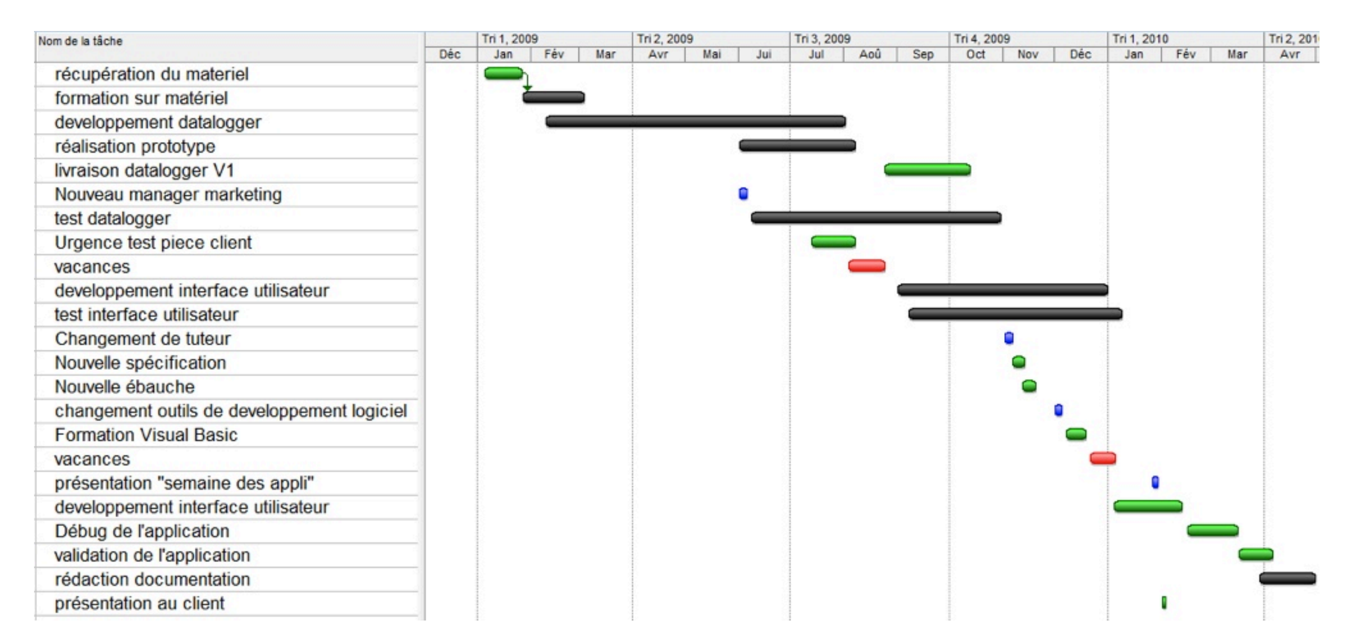
### 1.2. Economique

L'investissement général réservé à ce projet à été dépassé. Sur le plan matériel il a fallu faire appel au sous-traitant une seconde fois pour réaliser le datalogger en version boitier plastique, ce qui implique une dépense supplémentaire que je chiffrerais aux alentours de 500€. Sur le plan de rémunération et de gestion, l'allongement de la durée initiale du projet a engendré environ 21000€ supplémentaires. Ces coûts ne sont pas perçus comme des pertes mais des investissements puisque le projet dépasse aujourd'hui les attentes des premières spécifications.

Le retour sur investissement de ce projet est difficilement quantifiable, il faudrait connaitre quel client à acheté notre puce grâce à la démonstration du datalogger et en quelle quantité.

## 2. Méthode et gestion de projet

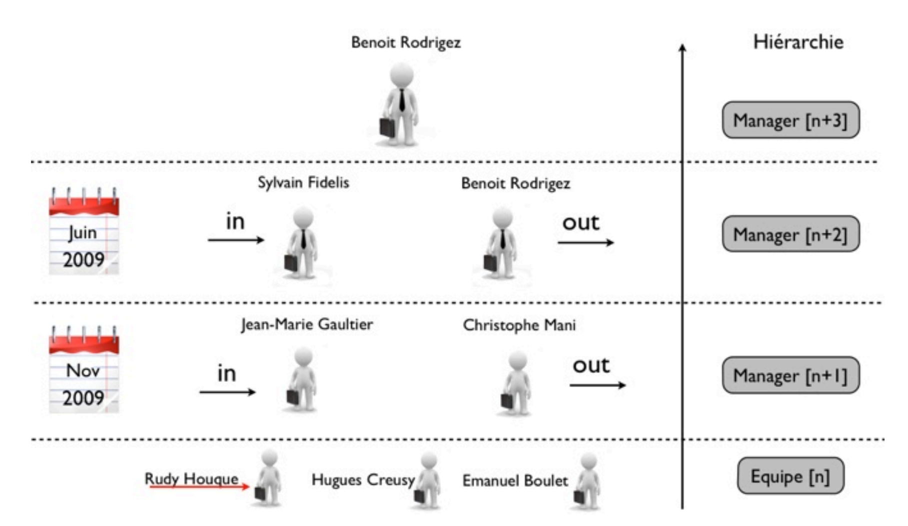
### 2.1. Planning réel



### 2.1.1. Imprévus :

**Réorganisation :**

En moins de quatre mois j'ai vu arriver un nouveau manager marketing et mon tuteur changer de service. Ces deux changements ont eu un impact majeur sur la façon de fonctionner de l'unité RFID.
Mes nouveaux supérieurs ont une vision «nouvelle» des projets en développement dans l'unité.




Ils décident de me faire modifier l'interface utilisateur alors même que le développement est déjà bien avancé.

Mon client étant ma hiérarchie directe, nous revoyons ensemble les nouvelles spécifications et dessinons une ébauche. Il est aussi question de faire entrer mon projet dans le projet d'un collègue en cours de développement.
Pour ce faire il me faut changer de langage de programmation afin que mon logiciel soit intégrable dans le sien. Un premier retard conséquent est visible à la suite de cette nouvelle car je ne connaissais pas le langage et j'ai du suivre une formation. Le développement l'interface utilisateur initialement prévue pour débuter fin aout est donc décalé de 4 mois.

**Urgences** :

Un client très important a connu un problème sur une de nos pièces courant juillet 2009 et en période estivale les effectifs de STMicroelectronics ne sont pas complets. J'ai donc du mettre de coté la réalisation de mon projet pendant prés d'un mois pour l'aider à résoudre ce problème.

**Les tâches additionnelles** :

La mémoire M24LR64 étant toute récente, elle n'est au départ produite qu'en version «prototype» en petite quantité et leur boitier est assemblé à la main. On compte environ deux mille pièces de la sorte qui sont envoyées à nos plus gros clients pour qu'ils commencent à développer leurs applications et ainsi qu'ils nous fassent remonter les difficultés qu'il rencontrent. Elles sont également envoyées aux constructeurs de lecteur RFID partenaires pour qu'ils puissent être compatibles avec cette nouvelle puce.

Les méthodes de test industrialisé n'étant pas encore en place pour ce nouveau produit, il faut les tester à la main. Je suis réquisitionné pour faire la partie RF de ces tests, ce qui me prend énormément de temps. C'est une tâche répétitive qui consiste à faire des essais de lecture et d'écriture à certaines distances de fonctionnement. Après avoir testé une centaine de pièces manuellement en une demi-journée, je décide de développer un logiciel permettant d'automatiser la méthode de test et de sortir un fichier de résultats, ce qui rend cette méthode beaucoup plus accessible. J'ai ensuite formé un opérateur sur le logiciel et lui ai délégué le test et la validation des puces. Cette solution m'a permis de me recentrer sur la réalisation de mon projet sans perdre trop de temps.

# 3. Avenir du projet

Le datalogger est une application terminée, il n'y a aucun projet de développement supplémentaire à court terme, néanmoins les couches logicielles développées pour cette application sont disponibles sur internet et réutilisables. Elles feront donc gagner du temps aux futurs développeurs lorsqu'ils réaliseront leurs propres dataloggers.
Le réel avenir du datalogger réside dans le support client autour de ce projet, il consiste à les aider à intégrer cette solution dans leurs applications.

# VII.Résultats

L'application datalogger sera disponible sur le site internet www.st.com avec sa documentation dans quelques jours elle est aujourd'hui uniquement sur notre site intranet.

Dual Interface EEPROM • User Manuals

| Title | Filename | Last Update | Contact | Info |
|---|---|---|---|---|
| Using the M24LR64-R datalogger reference design NEW | UM0925.PDF | 4-May-2010 | jean-marie gaultier | |
| M24LR64-R tool driver install guide | UM0863.PDF | 3-Dec-2009 | Pascal Castanet | |
| M24LR64-R tool kit user guide | UM0853.PDF | 29-Jan-2010 | Pascal Castanet | |
| Using your DEMOKIT-M24LR-A demonstration kit with your STM32-PRIMER2 | UM0850.PDF | 29-Jan-2010 | Pascal Castanet | |

**La semaine des applications**
Le datalogger a été présenté à l'ensemble du personnel STMicroelectronics à l'occasion de l'événement «la semaine des applications» en janvier 2010. Les personnes de l'équipe se sont relayées pour animer le stand réservé à la RFID et présenter les applications du service (voir photo ci dessous).

**Le salon «RFID Journal Live»**

Le manager marketing de l'équipe RFID Sylvain.F est allé présenter le datalogger au salon «RFID Journal Live» à Orlando en Floride (état-Unis) du 14 au 16 Avril 2010. L'application a remporté un grand succès.




«RFID Journal Live» est la plus grande manifestation internationale dédié aux technologies RFID.

**embedded systems conference**

Le data logger a aussi été présenté lors de la conference «embedded systems» à San Jose en Californie (état-Unis) du 26 au 29 Avril 2010. Encore une fois le datalogger a suscité un grand intérêt.







Le produit M24LR64 connait un certain succès en début de vie puisque la consommation annuelle prévisionnelle est déjà annoncée à plus d'un millions de pièces.

# VIII.Conclusion

J'ai fait le choix de poursuivre mon cursus scolaire en apprentissage afin d'allier connaissances théoriques et expérience professionnelle. Je suis aujourd'hui apprenti depuis presque trois années dans la société STMicroelectronics où j'ai pu effectuer ce projet de fin d'étude qui est en total adéquation avec l'intitulé de mon diplôme puisqu'il est composé d'une partie électronique et d'une partie informatique industrielle.

La réalisation du datalogger de température, m'a permis d'approfondir mes connaissances de la technologie et des produits RFID. La collaboration et les échanges quotidiens avec les ingénieurs de l'équipe m'ont énormément apporté, comme par exemple l'apprentissage de deux nouveaux langages de programmation (LabWindows-CVI et Visual Basic). Les cours que j'ai suivi en parallèle à l'école durant ces trois années ont été d'une aide précieuse pour l'approche technique de ce projet.

Ce projet et cette formation sont une expérience très enrichissante intellectuellement et personnellement. Contrairement aux périodes de stage relativement courtes que j'ai pu effectuer précédemment, l'apprentissage permet de faire partie intégrante d'une équipe, de se voir confier plus de responsabilités, et d'avoir un travail diversifié. Avec le recul, je me rends compte de l'importance de la communication au sein d'une entreprise, elle joue un des rôles les plus important dans la réussite de ce projet.

Pour remplir ma mission d'ingénieur d'application RFID, j'ai été amené à entrer en contact avec des clients extérieurs qui sont pour la grande majorité étrangers. Si l'idée même de tenir un discours professionnel en anglais m'effrayais en début de formation, je suis aujourd'hui capable de m'entretenir sur des sujets techniques, de comprendre et me faire comprendre.

Je sais également aujourd'hui grâce à cette formation dans quel domaine je souhaite exercer mon métier d'ingénieur, et c'est celui de la RFID. Je crois énormément en l'avenir des applications utilisant cette technologie et notamment pour l'internet des objets*. Mes trois années d'expérience professionnelle, la pratique de l'anglais quasi quotidienne s'ajoutant aux cours dispensés par les enseignants de l'école m'ont dores et déjà permis de me vendre plus facilement lors d'entretien d'embauche que j'ai pu passé depuis quelques mois et dont les retours sont très encourageants.

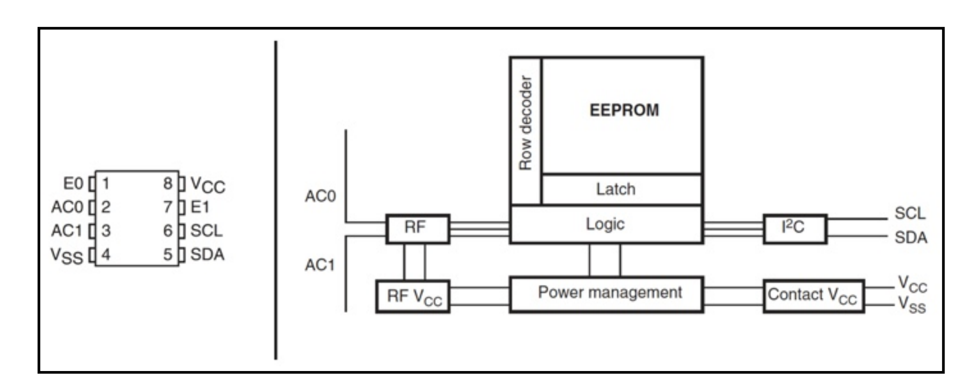
l'internet des objets* : représente l'extension d'Internet à des choses et à des lieux dans le monde réel.





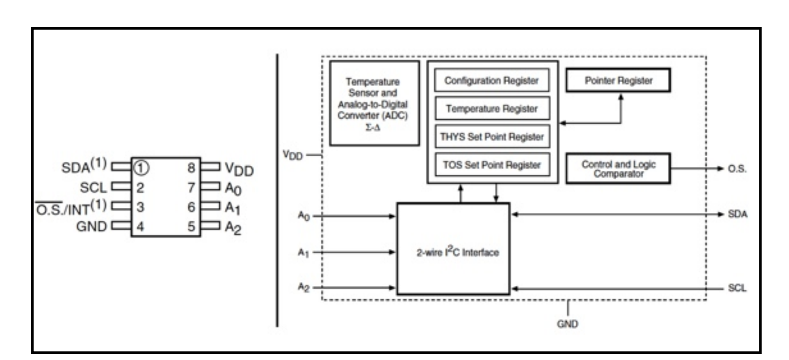
# IX.Annexe

## 1. Mémoire double interface M24LR64




*38.figure : diagramme block + pinout du M24LR64*

## 2. Capteur de Température STTS75

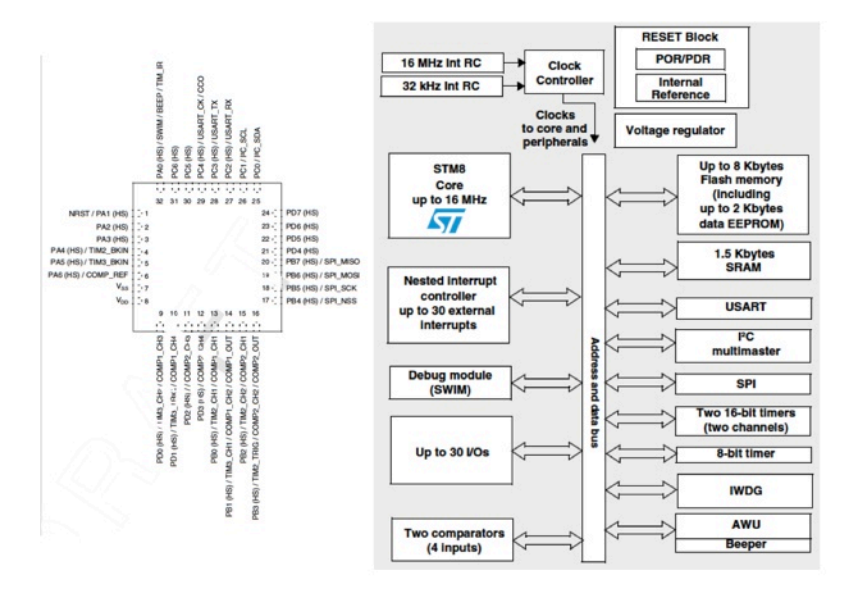



*39.figure : Diagramme block + pinout du STTS75*

| Temperature | Sign | Number of bits used by conversion resolution | | 9 | 10 | 11 | 12 | Always zero | Digital output (HEX) |
|---|---|---|---|---|---|---|---|---|---|
| | 12-bit resolution | | | | | | | 0000 | |
| | 11-bit resolution | | | | | | 0 | 0000 | |
| | 10-bit resolution | | | | | 0 | 0 | 0000 | |
| | 9-bit resolution | | | | 0 | 0 | 0 | 0000 | |
| +125°C | 0 | 111 | 1101 | 0 | 0 | 0 | 0 | 0000 | 7D00 |
| +25.0625°C | 0 | 001 | 1001 | 0 | 0 | 0 | 1 | 0000 | 1910 |
| +10.125°C | 0 | 000 | 1010 | 0 | 0 | 1 | 0 | 0000 | 0A20 |
| +0.5°C | 0 | 000 | 0000 | 1 | 0 | 0 | 0 | 0000 | 0080 |
| 0°C | 0 | 000 | 0000 | 0 | 0 | 0 | 0 | 0000 | 0000 |
| –0.5°C | 1 | 111 | 1111 | 1 | 0 | 0 | 0 | 0000 | FF80 |
| –10.25°C | 1 | 111 | 0101 | 1 | 1 | 1 | 0 | 0000 | F5E0 |
| –25.0625°C | 1 | 110 | 0110 | 1 | 1 | 1 | 1 | 0000 | E6F0 |
| –55°C | 1 | 100 | 1001 | 0 | 0 | 0 | 0 | 0000 | C900 |

*40.figure : Tableau de format température du STTS75*

## 3. Microcontroleur STM8L

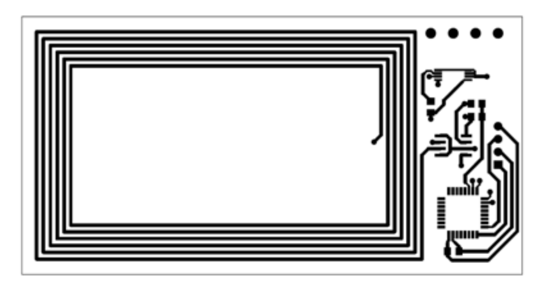



*41.figure : Diagramme block + pinout du STM8L*

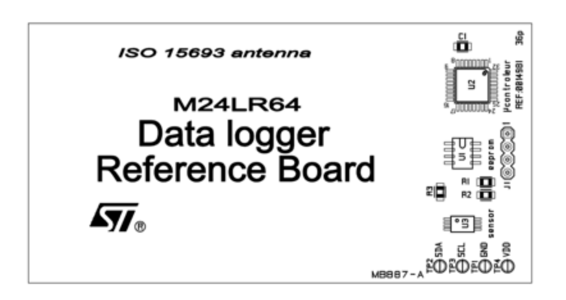
## 4. Schéma d'implantation carte datalogger V(1.0)



*42.figure :Carte datalogger (V1.0) routage face arrière.*

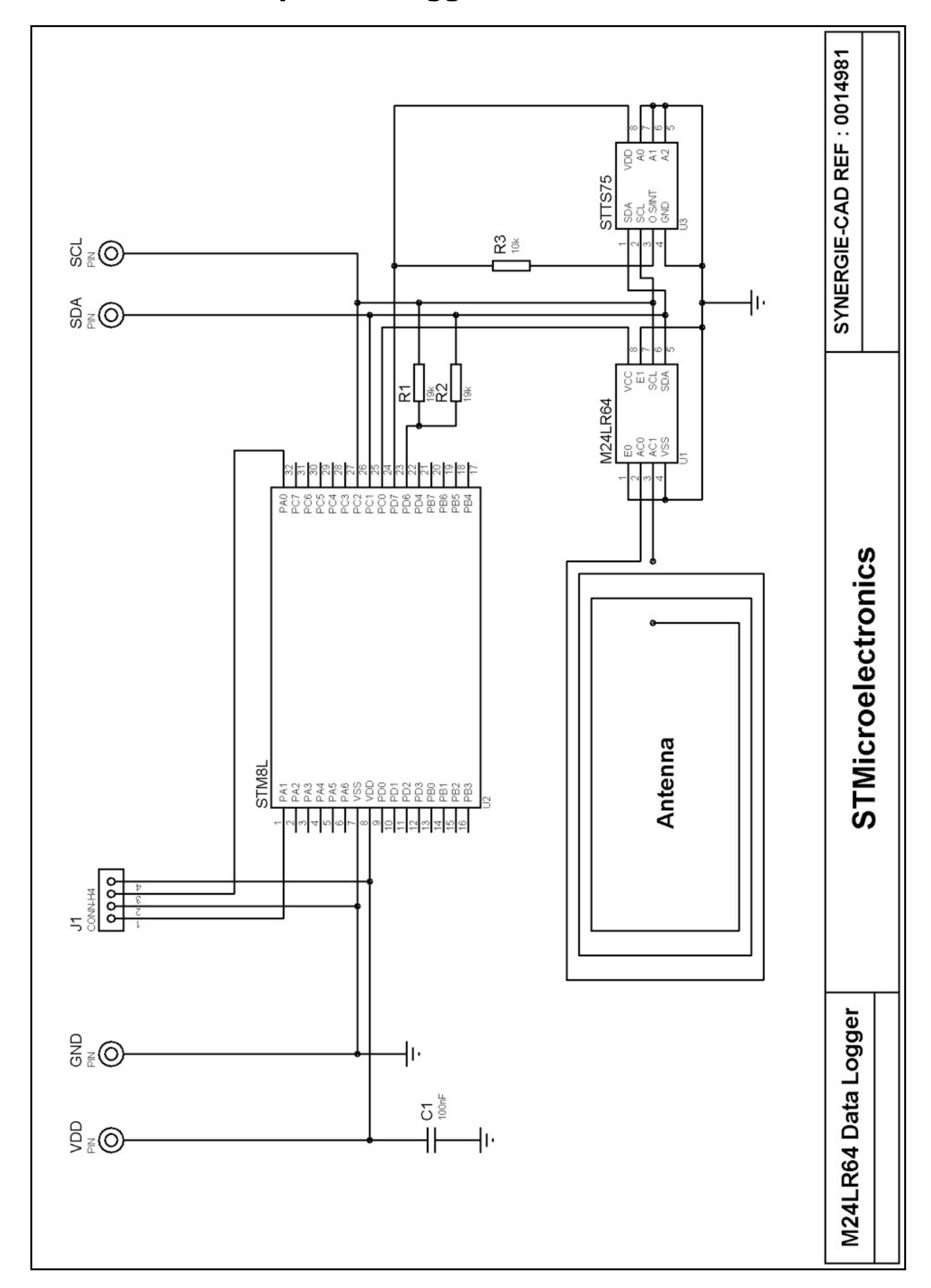


*43.figure : Carte datalogger (V1.0) routage face avant.*



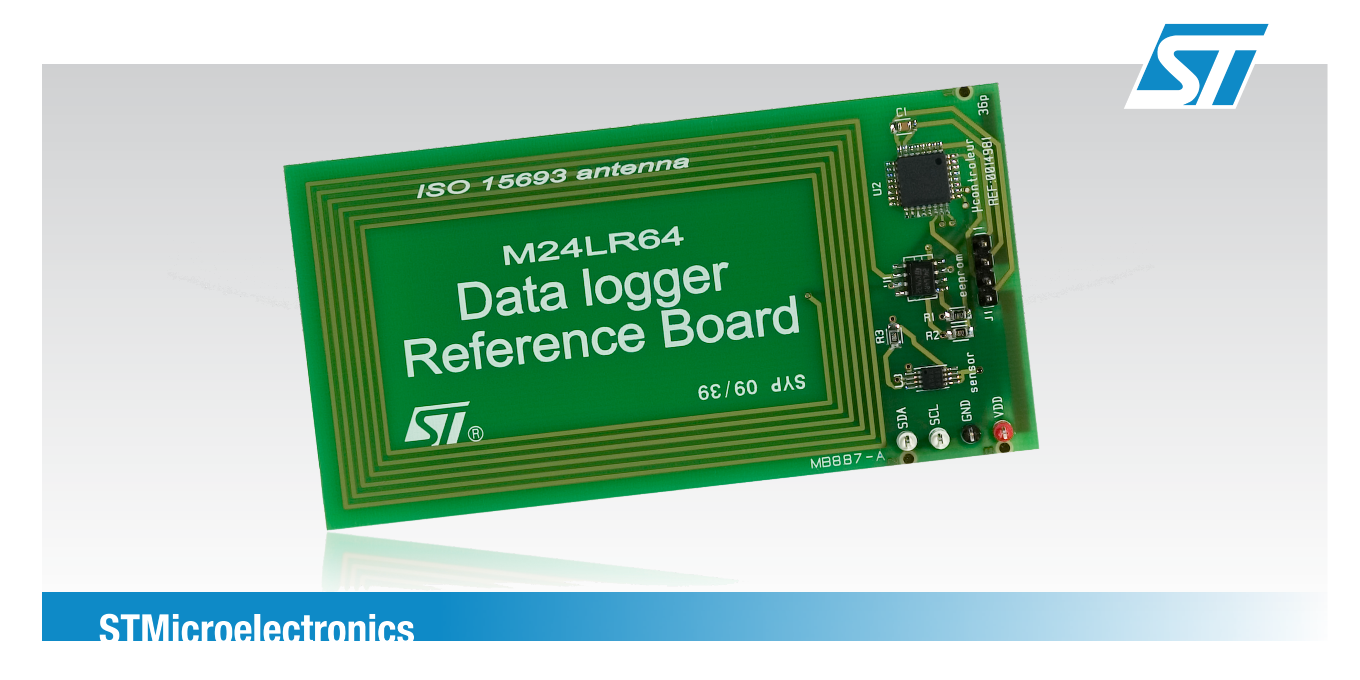

*44.figure : Carte datalogger (V1.0) sérigraphie avant.*

# 5. Schéma électrique datalogger

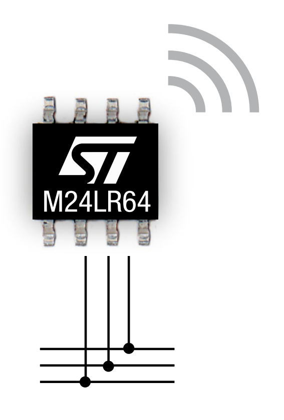


## 6. Affiche de promotion officielle

# M24LR64 FOR RFID-ENABLED DATALOGGERS




## Dual Interface 64-Kbit EEPROM with I²C and ISO 15693, a perfect fit for smart sensors




- ISO 15693 interface: provides a zero-power RF port
- I²C bus: connects to most microcontrollers (including STM8L)
- Low voltage operations: 1.8 to 5.5 V at 400 kHz
- 64-Kbit EEPROM: storage for large amount of data
- 32-bit password mechanism: data protection
- Available in a wide range of packages including wafer form
- High-reliability EEPROM memory

www.st.com

# 7. STM8L Source Code «main.c»

```c
/* Includes ------------------------------------------------------------------*/
#include "stm8l10x.h"
#include "stm8l10x_clk.h"
#include "stm8l10x_gpio.h"
#include "stm8l10x_awu.h"
#include "stm8l10x_tim2.h"
#include "i2c_ee.h"
#include "stm8l10x_i2c.h"
#include "stdio.h"
#include "stdlib.h"

/* Private defines -----------------------------------------------------------*/
#define BUFFER_SIZE ((uint8_t)2)
#define START   0x11
#define PAUSED  0x22
#define RUNNING 0x33
#define STOPPED 0x44
#define UPDATE  0x55

/* Private function prototypes -----------------------------------------------*/
void Delay(uint16_t nCount);
void AWU_AutoLSICalibration(void);

void start_acquisition   (void);
void stop_acquisition    (void);
void acquisition_running (void);
void acquisition_update  (void);

/* Global variable -----------------------------------------------------------*/
uint8_t  Status         = 0x00;                    /* SYSTEM BYTE */
uint8_t  Delay_Sleep    = 0x11;                    /* SYSTEM BYTE */
uint8_t  OverWrite      = 0x00;                    /* SYSTEM BYTE */
uint8_t  Nb_Temperature[2] = {0x00,0x00};  /* SYSTEM BYTE */

uint16_t FREE_SPACE     = 0x0FFC; /* 4092 free places to save temperatures */
uint16_t POINTER_ACTUAL = 0x0008; /* place to write the first temperature */
uint16_t Status_Add     = 0x0000; /* address of the Status Byte */
uint16_t OverWrite_Add  = 0x0001; /* address of the Overwrite Byte */
uint16_t Delay_Add      = 0x0002;  /* address of the Delay Byte */
uint16_t Nb_Temp_Add    = 0x0004; /* address of the Nb_Temp Bytes */

/*****************************************************************************/
/*------------------------------------- MAIN -----------------------------------------*/
/*****************************************************************************/
void main(void)
{
      /*----------------------  CLOCKS CONFIG ----------------------*/
      CLK_DeInit();/*all*/

      CLK_MasterPrescalerConfig(CLK_MasterPrescaler_HSIDiv8);/*2MHz*/
      CLK_PeripheralClockConfig(CLK_Peripheral_I2C, ENABLE);/*I2C*/
      CLK_PeripheralClockConfig(CLK_Peripheral_TIM2,ENABLE);/*AWU*/
      CLK_PeripheralClockConfig(CLK_Peripheral_AWU, ENABLE);/*AWU*/

      /*----------------------------  GPIO -------------------------*/
      /* Initialization of I/Os in Input Mode to minimize comsumption*/
      GPIO_Init(GPIOA, (GPIO_Pin_1 |GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5|
                           GPIO_Pin_6),GPIO_Mode_In_PU_No_IT );
      GPIO_Init(GPIOB, (GPIO_Pin_0 |GPIO_Pin_1 |GPIO_Pin_2|GPIO_Pin_3 | GPIO_Pin_4 |
                           GPIO_Pin_5 | GPIO_Pin_6| GPIO_Pin_7),GPIO_Mode_In_PU_No_IT );
      GPIO_Init(GPIOC, (GPIO_Pin_2 | GPIO_Pin_3| GPIO_Pin_4| GPIO_Pin_5|
      GPIO_Pin_6),GPIO_Mode_In_PU_No_IT );
      GPIO_Init(GPIOD, (GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3|
                           GPIO_Pin_4 ),GPIO_Mode_In_PU_No_IT );

      /* Initialization of I/Os in Output Mode */
      /* In order To power dual interface memory & temperature sensor & Bus i2c */
      GPIO_Init(GPIOD, GPIO_Pin_5| GPIO_Pin_6 |GPIO_Pin_7, GPIO_Mode_Out_PP_Low_Fast);

      AWU_AutoLSICalibration();   /* measure the LSI clock frequency */
      AWU_Init(AWU_Timebase_2s);  /* selection of time base */
      AWU_Cmd(ENABLE);            /* set the AWUEN bit */
      enableInterrupts();
```

```c
	while (1) /* routine start */
	{

		GPIO_SetBits(GPIOD, GPIO_Pin_5); /*power bus i2c*/
		GPIO_SetBits(GPIOD, GPIO_Pin_7); /*power eeprom*/

		if(Status == 0x00) /* first run */
		{
			stop_acquisition();
		}

		I2C_EEInit_EEPROM(); /* init eeprom dual*/
		I2C_EE_BufferRead(&Status, Status_Add, 0x01);/* read STATUS byte */

		if(Status == START)
		{
			start_acquisition();
		}

		else if(Status == RUNNING)
		{
			acquisition_running();
		}

		else if(Status == UPDATE)
		{
			acquisition_update();
		}

		GPIO_ResetBits(GPIOD, GPIO_Pin_6);/*Unpower sensor*/
		GPIO_ResetBits(GPIOD, GPIO_Pin_7);/*Unpower eeprom*/
		GPIO_ResetBits(GPIOD, GPIO_Pin_5);/*Unpower bus i2c*/

		halt();/*SLEEP*/

	}
}
```

## Fonction start_acquisition

```c
void start_acquisition(void)
{
	/* reset system variable */
	POINTER_ACTUAL     = 0x0008;
	Nb_Temperature[0] = 0x00;
	Nb_Temperature[1] = 0x00;
	FREE_SPACE         = 0x0FFC;
	Status             = RUNNING;

	I2C_EEInit_EEPROM(); /* init eeprom dual*/
	I2C_EE_BufferRead(&Delay_Sleep, Delay_Add, 0x01);/* READ DELAY */

	AWU_Init((uint8_t)Delay_Sleep); /* selection of time base */
	AWU_Cmd(ENABLE);                /* set the AWUEN bit */

	I2C_EE_PageWrite(&Status, Status_Add, 0x01);
	Delay((uint16_t)2000);

	I2C_EE_PageWrite(Nb_Temperature, Nb_Temp_Add, 0x02);
	Delay((uint16_t)2000);
}
```

# 8. STM8L Source Code «i2c_ee.c»

Fonction I2C_EE_PageWrite

```
void I2C_EE_PageWrite(uint8_t* pBuffer, uint16_t WriteAddr, uint8_t NumByteToWrite)
{
  /* While the bus is busy */
  while(I2C_GetFlagStatus(I2C_FLAG_BUSY));

  /* Send START condition */
  I2C_GenerateSTART(ENABLE);

  /* Test on EV5 and clear it */
  while(!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

  /* Send EEPROM address for write */
  I2C_Send7bitAddress(EEPROM_ADDRESS, I2C_Direction_Transmitter);

  /* Test on EV6 and clear it */
  while(!I2C_CheckEvent(I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
  I2C_GetFlagStatus(I2C_FLAG_ADDR);
      (void)(I2C->SR3);

  /* Send Address (on 2 bytes) of first byte to be written & wait event detection */
  I2C_SendData((uint8_t)(WriteAddr >> 8)); /* MSB */
  /* Test on EV8 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTING));
  I2C_SendData((uint8_t)(WriteAddr)); /* LSB */
  /* Test on EV8 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTING));


  /* While there is data to be written */
  while(NumByteToWrite--)
  {
    /* Send the current byte */
    I2C_SendData(*pBuffer);

    /* Point to the next byte to be written */
    pBuffer++;

    /* Test on EV8 and clear it */
    while (!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));
  }

  /* Send STOP condition */
  I2C_GenerateSTOP(ENABLE);
}
```





## Fonction I2C_SS_BufferRead

```
void I2C_SS_BufferRead(uint8_t* pBuffer, uint8_t Pointer_Byte, uint8_t NumByteToRead)
{
	/* While the bus is busy */
  while(I2C_GetFlagStatus(I2C_FLAG_BUSY));

  /* Generate start & wait event detection */
  I2C_GenerateSTART(ENABLE);
  /* Test on EV5 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

  /* Send slave Address in write direction & wait detection event */
  I2C_Send7bitAddress(SENSOR_ADDRESS, I2C_Direction_Transmitter);
  /* Test on EV6 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
  I2C_GetFlagStatus(I2C_FLAG_ADDR);
	(void)(I2C->SR3);

  /* Configure the sensor in read mode Pointer 0x00 */
  I2C_SendData(Pointer_Byte); /* MSB */
  /* Test on EV8 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));

  /* Send START condition a second time */
  I2C_GenerateSTART(ENABLE);
  /* Test on EV5 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

  /* Send slave Address in read direction & wait event */
  I2C_Send7bitAddress(SENSOR_ADDRESS, I2C_Direction_Receiver);
  /* Test on EV6 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
  I2C_GetFlagStatus(I2C_FLAG_ADDR);
	(void)(I2C->SR3);

  /* While there is data to be read */
  while(NumByteToRead)
  {
    if(NumByteToRead == 1)
    {
      /* Disable Acknowledgement */
      I2C_AcknowledgeConfig(DISABLE);
		}

    /* Test on EV7 and clear it */
    if(I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_RECEIVED))
    {
      /* Read a byte from the SENSOR */
      *pBuffer = I2C_ReceiveData();

      /* Point to the next location where the byte read will be saved */
      pBuffer++;

      /* Decrement the read bytes counter */
      NumByteToRead--;
    }
    if(NumByteToRead == 0)
    {
      /* Send STOP Condition */
      I2C_GenerateSTOP(ENABLE);
		}
	}

  /* Enable Acknowledgement to be ready for another reception */
	I2C_AckPositionConfig(I2C_AckPosition_Current);
}
```

# 9. Visual Basic source code

Fonction start_acquisition_Click

```
'*************************************************
'************** START BUTTON *******************
'*************************************************
Private Sub start_acquisition_Click()

Dim Read_Single_Result As String
Dim Status_Byte As String
Dim Delay_Byte As String
Dim Data_To_Send As String
Dim Txt_Message_Box As String
Dim iShowStopButton As Integer

iShowStopButton = 0
meteo_picture.Visible = True
RealTime.Visible = False
MSChart1.Visible = False
TXTDynamicview.Visible = False
iAnimation = ANIM_NOTHING
iBattery = ENOUGH
iRealTimeGraph = PERMITED

'******************** INVENTORY ***********************
If (Inventory_DataLogger = SUCCEDED) Then
      board_picture = LoadPicture(App.Path & "\images\DataLogger_Pictures\RF.bmp")
      iAnimation = ANIM_RF

      '***************** READ BLOCK 0000 *****************
      Read_Single_Result = ReadRF_single_DataLogger(0, 4, 2)
      If (iReadRF_Success = SUCCEDED) Then
                Data_To_Send = Read_Single_Result
                Status_Byte = Mid(Read_Single_Result, 1, 2) 'get the system status
                Delay_Byte = Mid(Read_Single_Result, 5, 2) 'get the delay value in order to authorized the
realtime graph view

                If Status_Byte = START Then
                        iBattery = LOW
                MsgBox "The battery may be low, please try to turn the data logger
                                  OFF and ON again", _vbInformation, _"Information window"

           ElseIf Status_Byte = PAUSED Then
                 Mid(Data_To_Send, 1, 2) = START
                 iShowStopButton = 1

           ElseIf Status_Byte = RUNNING Then
                iShowStopButton = 1

           ElseIf Status_Byte = STOPPED Then
                Mid(Data_To_Send, 1, 2) = START
                iShowStopButton = 1

           Else
                Mid(Data_To_Send, 1, 2) = START
                iShowStopButton = 1

           End If
        End If
      '--------------- END READ BLOCK 0000 -------------------

      If (Delay_Byte = "0D") Then
                iRealTimeGraph = PERMITED
      Else
                iRealTimeGraph = NON_PERMITED
      End If

      '***************** WRITE BLOCK 0000 ********************
      If (WriteSingleBlockRF_DataLogger(0, Data_To_Send, 4, 2) = SUCCEDED) And (iBattery <> LOW) Then
      TimerAnimation.Enabled = True 'start the board Animation
      TimerThermometre.Enabled = True 'start the meteos application
      iAnimation = ANIM_I2C 'kind of animation to play
                If (iShowStopButton = 1) Then 'change the start button in stop button
                        start_acquisition.Visible = False
                        TXTstart.Visible = False
                        stop_acquisition.Visible = True
                        TXTstop.Visible = True
                End If
      ElseIf (iBattery = LOW) Then
                board_picture = LoadPicture(App.Path & "\images\DataLogger_Picture \LOW.bmp")
      Else
                iAnimation = ANIM_NORF 'kind of animation to play
                MsgBox "Start condition has not been sent !! Please retry", _vbCritical, _
                        "Information window"
      End If
```





```
    '***************** END WRITE BLOCK 0000 ****************

      Else
                board_picture = LoadPicture(App.Path & "\images\DataLogger_Pictures
                                                                 \NORF.bmp")
      End If
    '******************* END INVENTORY ***********************

End Sub
```

Fonction ReadRF_single_DataLogger

```
'-----------------------------------------------------------------------------
'------------------------- ReadRF_single_DataLogger -------------------------
'-----------------------------------------------------------------------------
' the string contains the four read bytes or one of the folowing expressions
' "no_detected_answer"
' "no_tag_answer "
'-----------------------------------------------------------------------------
' Example : sResult = ReadRF_single_DataLogger(0, 4, 2)
' means that you will read 4 bytes at the address 0 coded on two bytes (@0000)
'-----------------------------------------------------------------------------
Function ReadRF_single_DataLogger(lngAddLow As Long, lngDataSize As Long, lngNbByteAddress As Long) As String
Dim strRequestDatas As String
Dim lngRequestDatasLen As Long
Dim lngRspLength As Long
Dim strResponseDatas As String * 512
Dim lngResponseDatasLen As Long
Dim bytRequestFlag As Byte
Dim strRequestFlag As String

Dim iDataRate  As Integer

Dim strResponseBlocking As String
Dim lngadd As Long
Dim strTmpData As String
Dim strTmpSSS As String
Dim strAddDisplay As String

Dim strAddress As String

Dim boolLoop As Boolean

Dim i, j As Long
Dim NbRespDatas As Long
Dim Shift As Long
Dim lngStatus As Long
Dim lngMode As Long
Dim bytOrganization As Long


'Estar
Dim lngEstarCmdSize As Long
Dim pEstarCmd(0 To 63) As Byte
Dim pEstarAnswerSize(0 To 63) As Byte
Dim pEstarAnswer(0 To 63) As Byte

For iPolling = 0 To NB_RETRY

      'The function is considered as FAILED the status only changes in case of SUCCESS
      iReadRF_Success = FAILED
      bytOrganization = 4


      lngRspLength = &H38 'response lenght (Feig USB Reader)

      'REQUEST FLAG MANAGEMENT + Extended Flag Option FOR DUALMODE
     bytRequestFlag = gbytRequestFlag
     bytRequestFlag = bytRequestFlag And &HDF 'Address Flag = 0
     bytRequestFlag = bytRequestFlag And &HEF 'Select Flag = 0

     strRequestFlag = i2hhh(CLng(10), 2)

     lngadd = lngAddLow

     strAddress = i2hhh(lngadd And &HFF, 2)
     strAddress = strAddress & i2hhh((lngadd \ (2 ^ 8)) And &HFF, 2)

     strRequestDatas = strRequestFlag & _cstrCmdReadSingleBlock & _strAddress

     strRequestDatas = Replace(strRequestDatas, " ", "")
     lngRequestDatasLen = Len(strRequestDatas)

        'READER SELECTION
       Select Case giSelectedReader_RF
```

```
            Case cReader_RF_Feig_USB, cReader_RF_Feig_RS232 'FEIG Reader
            strRequestDatas = gstr0xBFcmd_Reserved_7_8 & strRequestDatas
            lngRequestDatasLen = Len(strRequestDatas)
            lngStatus = FEISC_0xBF_ISOTranspCmd(lngAttachedDeviceHandle(1), &HFF,
                                  &H1, lngRspLength, strRequestDatas, lngRequestDatasLen,
                                  strResponseDatas,lngResponseDatasLen, 1)

            Case cReader_RF_eStar_USB 'ESTART READER
            strResponseDatas = ""
            lngRequestDatasLen = Len(strRequestDatas) / 2
            strRequestDatas = gstr_estar_0xB0FF_cmd & i2hhh(lngRequestDatasLen, 2) &
                                  strRequestDatas
            lngEstarCmdSize = Len(strRequestDatas) / 2
            For j = 0 To lngEstarCmdSize - 1
                            pEstarCmd(j) = CByte("&h" & Mid(strRequestDatas, (j * 2) + 1, 2))
            Next j

                    lngStatus = API_USBAll(hcomm_public, lngEstarCmdSize, pEstarCmd(0),
                            pEstarAnswerSize(0), pEstarAnswer(0))
            If (lngStatus = 4) Then
                    strDataMsgEssai = "The software lost the communication with
                                                    the eStar reader." & vbCrLf & _"Please,
                                                    close the sofware window." & vbCrLf &
                                                    _"Disconnect the eStar reader and connect
                                                    it again." & vbCrLf & _"You will be able
                                                    to launch again the application."

                    Call MsgBox(strDataMsgEssai, vbInformation, "eStar reader
                                    problem")

            End If

            For j = 0 To pEstarAnswerSize(0) - 1
                    Mid(strResponseDatas, (j * 2) + 1, 2) = i2hhh(CLng(pEstarAnswer(j)), 2)
          Next j

      End Select

      lngResponseDatasLen = Len(strResponseDatas)

      If (lngResponseDatasLen > 0) Then

            If (Mid(strResponseDatas, 1, 2) = "00") Then

                    strResponseBlocking = ""
                    strResponseDatas = Replace(Mid(strResponseDatas, 3,
                                    lngResponseDatasLen - 6), " ", "")

                    strTmpData = "" 'format

                    For j = 0 To lngDataSize - 1 'Step 2
                    strTmpData = strTmpData & Mid(strResponseDatas, 1 + j * 2, 2)
                    Next j

                    iReadRF_Success = SUCCEDED
                    iPolling = NB_RETRY

            ElseIf (Mid(strResponseDatas, 1, 2) = "01") Then
                    iReadRF_Success = FAILED
            Else
                    strTmpData = "no_tag_answer"
                    iReadRF_Success = FAILED
            End If
      Else
            strTmpData = "no_detected_answer"
            iReadRF_Success = FAILED
      End If

      DoEvents


      ReadRF_single_DataLogger = strTmpData

      Next iPolling

End Function
```